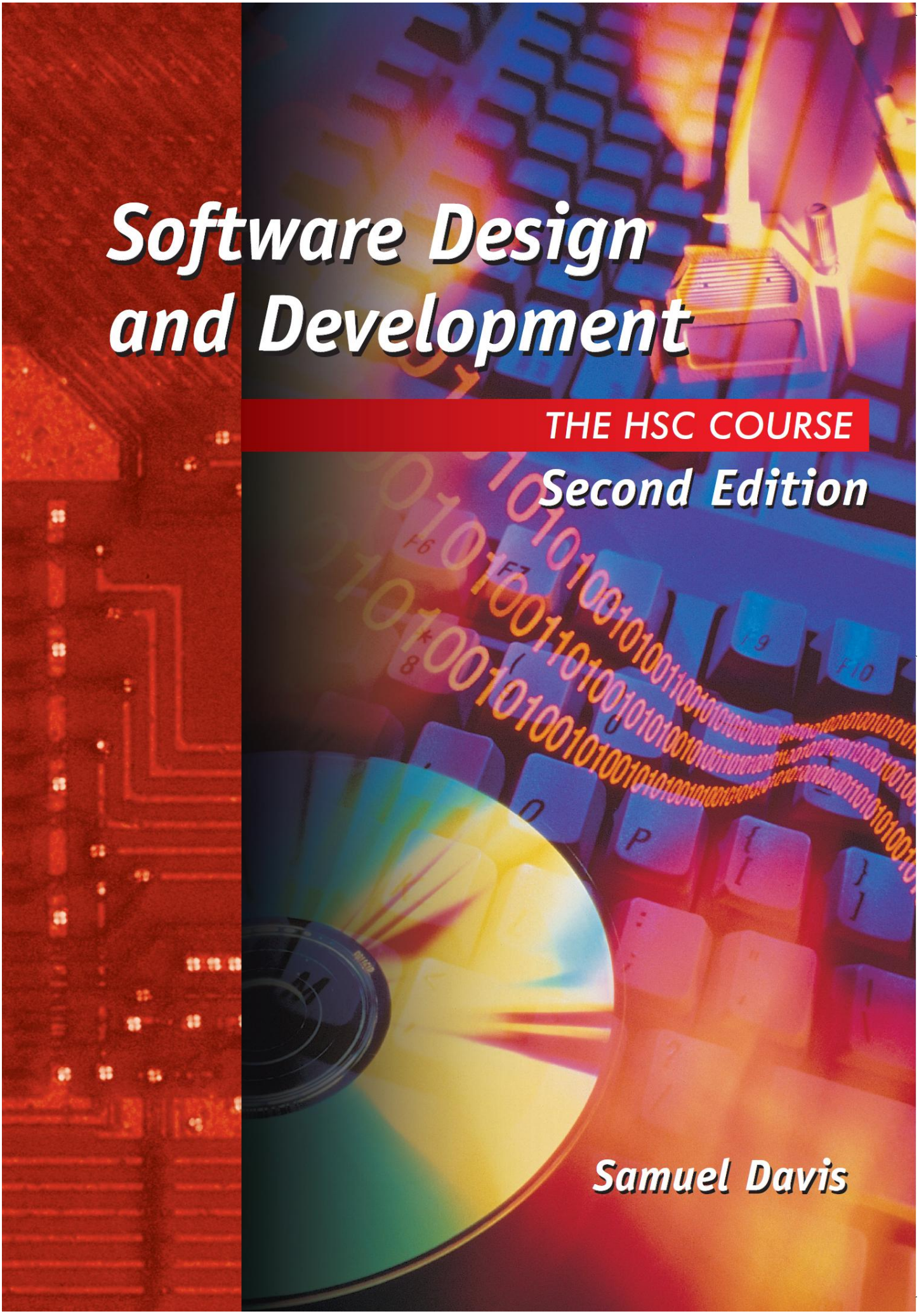
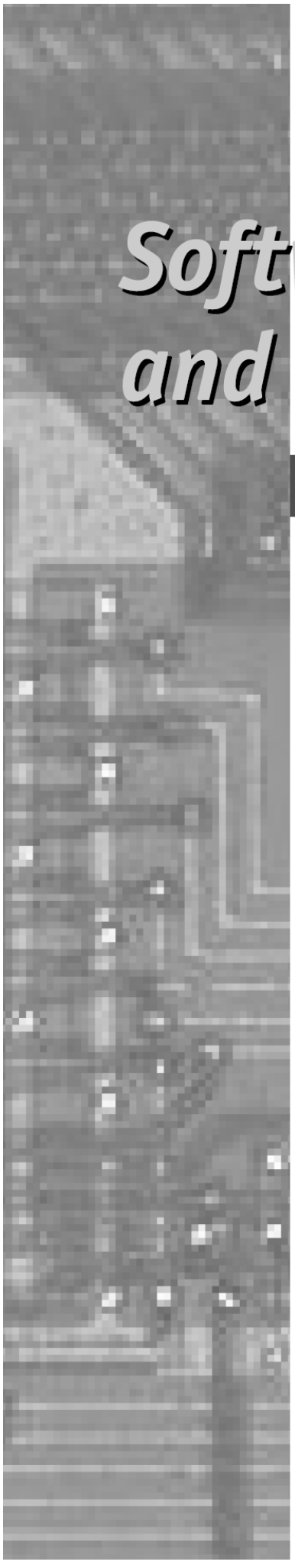

Software Design
and Development
THE HSC COURSE
Second Edition
Samuel Davis



Software Design and Development

THE HSC COURSE

Second Edition

Samuel Davis

Contributing Author
Janine Fendall

Parramatta Education Centre

First published 2012 by
Parramatta Education Centre
Tel: (02) 4632 7987 Fax: (02) 4632 8002

Visit our website at www.pedc.com.au

Copyright © Samuel Davis 2012

All rights reserved.
Except under the conditions described in the Australian Copyright Act 1968 (the Act) and subsequent amendments, no part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior permission of the copyright owner.

National Library of Australia
Cataloguing in publication data

Davis, Samuel, 1964-.
Software design and development: the HSC course (second edition).

Includes index.
ISBN 9780980874921.

1. Computer software – Development. I. Title.

005.3

Contributing editor: Janine Fendall
Cover design: Great Minds
Printed in Australia by Ligare Pty. Ltd.

CONTENTS

ACKNOWLEDGEMENTS viii

TO THE TEACHER ix

TO THE STUDENT ix

DEVELOPMENT AND IMPACT OF SOFTWARE SOLUTIONS

1. SOCIAL AND ETHICAL ISSUES 3

The impact of software 3

Rights and responsibilities of software developers 10
- Intellectual property 11
- Quality 12
- Response to problems 16
- Code of conduct 17
- Malware 18
- Ergonomic issues in software design 19
- Inclusivity issues 21
- Privacy issues 25
- **Set 1A 27**

Software piracy and copyright 28
- Plagiarism 28
- Copyright laws 29
- Classification of software in terms of copyright 30
- Ownership versus licensing 32
- Software licence agreements 32
- *HSC style question 36*
- Current and emerging technologies used to combat software piracy 36
- *HSC style question 41*
- **Set 1B 42**

Use of networks 44

The software market 46
- **Set 1C 50**

Legal implications 51
- National legal action 51
- International legal action 53
- **Chapter 1 review 56**

2. APPLICATION OF SOFTWARE DEVELOPMENT APPROACHES 59

Approaches used in commercial systems 59
- Structured approach 60
- Agile approach 62
- Prototyping 66
- RAD (rapid application development) 68
- End user development 72
- Combinations of software development approaches 74
- *HSC style question 75*
- **Set 2A 77**

CASE tools in large system development 78

Methods of installation 80
- *HSC style question 86*
- **Set 2B 87**

Current trends in software development 89
Employment trends 89
Trends in software development 92
Set 2C **96**
HSC style question *98*
Chapter 2 review **100**

SOFTWARE DEVELOPMENT CYCLE

3. DEFINING AND UNDERSTANDING THE PROBLEM 103

Defining the problem 103
Needs of the client 103
Boundaries of the problem 108
Issues relevant to a proposed solution 109
Selecting and appropriate development approach 113
HSC style question *114*
Set 3A **115**

Design specifications 117
Design specifications from the developer's perspective 117
Design specifications from the user's perspective 118

Communication issues 119
Set 3B **120**

System documentation 121

System modelling tools 121
System flowcharts 122
HSC style question *123*
Context diagrams 125
HSC style question *126*
Data flow diagrams 126
HSC style question *129*
Structure charts 130
HSC style question *133*
Input process output (IPO) diagrams 135
Data dictionary 137
Screen designs and concept prototypes 140
Storyboards 141
HSC style question *143*
Set 3C **144**

Quality assurance 145
HSC style question *147*
Chapter 3 review **148**

4. PLANNING AND DESIGNING SOFTWARE SOLUTIONS 151

Standard algorithms 153
Big O notation 153
Linear search 155
Finding maximum and minimum values 156
Binary search 158
HSC style question *160*
Set 4A **161**
Insertion sort 162
Selection sort 165
Bubble sort 168
HSC style question *172*
Set 4B **174**
Processing strings 175
Random numbers 177
HSC style question *181*
Set 4C **184**

Custom designed logic used in software solutions 185
Data structures and files 185
- Multi-dimensional arrays 185
- Arrays of records 189
- Files 192
HSC style question *195*
Set 4D **200**
Creating custom-designed algorithms 201
Set 4E **208**
Interface design in software solutions 210
Identification of data required 210
Current popular approaches 213
Design of help screens 214
Consistency in approach 216
Set 4F **217**
Customisation of existing software solutions 218
Selection of language to be used 219
Factors to be considered when selecting the technology to be used 221
Chapter 4 review **222**
5. IMPLEMENTATION OF SOFTWARE SOLUTIONS 225
Choosing an appropriate programming language 225
Language syntax required for software solutions 227
Use of EBNF and railroad diagrams 227
Metalanguage descriptions of programming languages 228
HSC style question *236*
Set 5A **238**
Translation methods in software solutions 240
Interpretation 241
Compilation 242
The translation process in detail 243
- Lexical analysis 243
- Syntactical analysis 245
- Code generation 247
HSC style question *248*
Set 5B **249**
The role of machine code in the execution of a program 250
What is machine code? 250
How is machine code executed by the CPU? 252
HSC style question *255*
Set 5C **257**
Techniques used in developing well written code 258
The use of good programming practices 259
Errors occurring in software solutions 263
Techniques for detecting errors in code 266
Software tools for detecting errors in code 271
HSC style question *274*
Set 5D **276**
Documentation of a software solution 278
User documentation 278
Technical documentation 282
HSC style question *286*
Set 5E **287**
Hardware environment to enable implementation of the software solution 288
Emerging technologies 289
Chapter 5 review **292**

6. TESTING AND EVALUATION OF SOFTWARE SOLUTIONS ____ 295

Comparison of the solution with the design specifications 296
How do we test the design specifications? 296

Levels of testing 297
What is system level testing? 297

Use of live test data for system testing 299
Large file sizes 299
Mix of transaction types 300
Response times 301
Volume data (load testing) 302
Interfaces between modules and programs 303
Comparison of actual with expected results 304

Benchmarking 304

Quality assurance 305
Set 6A **307**

CASE tools for testing software solutions 309

Reporting on the testing process 311
Documentation of the testing process 311
Communication of test results 312

Evaluating the software solution 314
Post implementation review 314
Acceptance testing 314
HSC style question *315*
Chapter 6 review **316**

7. MAINTAINING SOFTWARE SOLUTIONS _____ 319

Modification of code to meet changed requirements 320
Identification of the reasons for change in source code 320
Location of section to be altered 322
Determining changes to be made 324
Implementing and testing the solution 326
Set 7A **332**

Documenting changes 330
Source code documentation 331
Updating associated hardcopy documentation and online help 332
Use of CASE tools to monitor changes and versions 333
HSC style question *334*
Chapter 7 review **338**

SOFTWARE PROJECT

8. DEVELOPING A SOLUTION PACKAGE _____ 341

Designing and developing a software solution 342
Logbooks 342
Defining the problem 342
Understanding the problem 343
Planning and design of a solution 345

Implementation of a solution 348
Maintenance 351
HSC style question *352*
Chapter 8 review **357**

OPTIONS

9. OPTION 1 –PROGRAMMING PARADIGMS 361

Development of the different paradigms 362
- Limitations of the imperative paradigm 362
- Emerging technologies 364
- Introduction to different paradigms 365
- **Set 9A 369**

Paradigm specific concepts 371
- Logic paradigm 372
- *HSC style question 377*
- Appropriate use of the logic paradigm 383
- - Expert systems 384
- **Set 9B 386**
- Object oriented paradigm 388
- *HSC style question 400*
- **Set 9C 404**

Programmer productivity 406
- *HSC style question 410*
- **Chapter 9 review 412**

10. OPTION 2 – THE INTERRELATIONSHIP BETWEEN SOFTWARE AND HARDWARE 415

Representation of data within the computer 416
- The binary number system 416
- Character representation 421
- **Set 10A 426**
- Integer representation 427
- Representation of fractions and real numbers 430
- **Set 10B 434**
- Binary arithmetic 437
- *HSC style question 442*
- **Set 10C 445**

Electronic circuits to perform standard software operations 446
- Logic gates 446
- Circuit design 449
- Boolean algebra 451
- Conversion between truth tables, Boolean expressions and circuit diagrams 454
- *HSC style question 455*
- Specialty circuits 458
- - Binary adders 458
- **Set 10D 460**
- - Flip-flops as memory stores 462
- - Shift registers 466
- *HSC style question 469*
- **Set 10E 470**

Programming of hardware devices 471
- Input and output data streams from sensors and other devices. 471
- Processing of the data stream 473
- Control systems 476
- *HSC style question 477*
- **Set 10F 482**
- **Chapter 10 review 485**

GLOSSARY 487

INDEX 497

ACKNOWLEDGEMENTS

I wish to thank Janine Fendall for her contribution and support during the production of this text. Janine has provided the motivation and incentive required to complete a task such as this. Her intelligent comments and original ideas have greatly improved the final product.

Thanks to all the computer teachers who have made comments and suggestions, I hope I've successfully implemented these to your satisfaction. Thanks also to the hundreds of students over the years who have unknowingly been "guinea pigs" for many of the ideas and strategies in this text.

Thanks to the many companies who willingly assisted with the provision of screen shots and other copyrighted material. Every effort has been made to contact and trace the original source of copyright material in this book. I would be pleased to hear from copyright holders to rectify any errors or omissions.

Samuel Davis

TO THE TEACHER

This book is written to cover the revised NSW *Software Design and Development* HSC course first examined in 2012. The text is written to closely reflect the syllabus both in terms of content and intent. The book uses a large variety of practical and commercial examples to assist students to realise the application of concepts in professional software development environments.

New concepts are initially introduced using familiar scenarios. These concepts are then further developed in a software development context. It is hoped that this will improve student understanding of the underlying concepts. Understanding of fundamental concepts is central to any study of software systems. The concepts learnt can then be applied to a broad range of software development activities. Students should be given the chance to experience a variety of software development tools and languages. This will allow them to make intelligent and knowledgeable choices.

Students are required to complete a software development projects as part of their assessment. These projects should be ongoing throughout the HSC year, portions of the project being completed as topics in the book are completed.

Every effort has been made to include the most up-to-date information in this text. However computer technologies are changing almost by the minute, which makes the writing task somewhat difficult. Technologies that are emerging today will be common place tomorrow. For this reason no internet addresses or details in regard to specific hardware technologies are used in the text.

TO THE STUDENT

This text is written for you. It is intended to help you understand and enjoy the processes involved in the development of software products. Software development is a booming industry. Software development is still a relatively new area and as such many techniques and tools are rapidly changing. Nevertheless the underlying thought processes and concepts required to develop software remain stable. During your study of this course, it is important to concentrate on understanding the concepts as well as acquiring proficiency and skills with particular tools and languages.

This book covers the content of the revised *Software Design and Development* HSC course. You must complete all the core topics covered in chapters 1-7 and one of the options topics, chapter 9 “Programming Paradigms” or chapter 10 “The Interrelationship between Software and Hardware”. During this course you will complete one or more software development projects. Chapter 8 describes the development of a sample project. Select projects that are both achievable and of personal interest to you.

Best wishes with your *Software Design and Development* studies and the HSC in general. Many of you will no doubt gain employment in the field of software development. I hope completion of this course provides a solid grounding for future study at tertiary level.

In this chapter you will learn to:

- recognise the effects of software solutions on society
- identify the impact of inappropriately developed software on users
- identify the effect of the inappropriate use of software on society and individuals
- apply a relevant code of conduct to their own software development
- interpret licence agreements and develop personal practices that reflect current laws
- identify the relationship between copyright laws and software license agreements
- acknowledge all sources in recognition of the intellectual contribution of authors
- identify a range of techniques designed to combat software piracy
- evaluate the usefulness of networks in the development environment
- identify the impact of dominant developers of software on software development
- discuss the reasons for, and consequences of, significant legal actions pertaining to the development of software

Which will make you more able to:

- explain the relationship between emerging technologies and software development.
- identify and evaluate legal, social and ethical issues in a number of contexts.

In this chapter you will learn about:

The impact of software
- inappropriate data structures, for example the year 2000 problem
- computer malware such as viruses
- reliance on software
- social networking
- cyber safety
- huge amounts of information (which may be unsupported, unverifiable, misleading or incorrect) available through the internet

Rights and responsibilities of software developers
- acknowledging the intellectual property of others
- recognition by others of the developer's intellectual property
- producing quality software solutions
- appropriately responding to user-identified problems
- adhering to code of conduct
- neither generating nor transmitting malware
- addressing ergonomic issues in software design
- ensuring software addresses inclusivity issues
- ensuring individuals' privacy is not compromised

Software piracy and copyright
- concepts associated with piracy and copyright, including:
  - intellectual property
  - plagiarism
  - copyright laws
  - licensing issues
  - licence conditions
  - shareware
  - public domain
  - open source
  - ownership versus licensing
  - collaboratively developed software
  - reverse engineering
  - decompilation
- current and emerging technologies used to combat software piracy. Such as:
  - non-copyable data sheet
  - use of serial numbers
  - site licence installation counter on a network
  - encryption key
  - registration code
  - back-to-base authentication

Use of networks
- by the developer when developing software
  - access to resources
  - ease of communication
  - productivity
- by the user when using network based software
  - response times
  - interface design
  - privacy and security issues

The software market
- maintaining market position
- the effect of dominant developers of software
- the impact of new developers of software and new products

Legal implications
- national and international legal action resulting from software development. Identify issues raised in cases at both national and international level, such as:
  - RACV vs Unisys
  - Microsoft vs Netscape
  - NSW T Card system
  - search engines (e.g. Google vs national censorship laws)
  - Metallica vs Napster

1

SOCIAL AND ETHICAL ISSUES

The development of software solutions has a profound impact on the community and the wider world. Software developers have a responsibility to ensure their products function in a socially responsible and ethical manner. Many software products perform tasks that are vital to the operation of essential services. Many of these services rely on computer software to perform their tasks adequately.

The users of computer software also have responsibilities. To ensure the continued production of quality software products requires developers to include systems to minimise the chance of software piracy. Copyright laws are designed to protect intellectual property rights. Such laws provide developers with legal methods of pursuing piracy cases. Software developers use licence agreements in an attempt to enforce their intellectual property rights and to minimise their liability should the software cause unintended problems.

In this chapter, we examine the impact of software on society. This includes the rights and responsibilities of software developers. Software developers are authors; they produce original works and as such are entitled to have their intellectual property protected in the same manner afforded to any other author. Unfortunately, software products are on the whole easy products to copy. What's more, the copy is identical to the original. This has lead to widespread piracy of software products. We examine techniques available to developers to combat piracy and other breaches of copyright.



GROUP TASK Discussion
Do you think it is the responsibility of software developers to consider social and ethical issues. Wouldn't it be easier if they could just concentrate on the task of writing software? Discuss.

THE IMPACT OF SOFTWARE

Software increasingly underlies more and more aspects of our lives. It is used to control the provision of essential services such as electricity, gas and water. Virtually all communication systems including traditional phone lines are largely driven by software. Every message sent across the internet, including email, web access, financial transactions and even many phone conversations utilise a variety of software systems. Most physical products use software during their design and manufacture and most electrically powered products include embedded software. Clearly the effect of software on society is significant.

When software is inappropriately developed or used the impacts have the potential to be significant. Some problems are due to poorly developed software that includes bugs, other problems are intentional such as malware and other issues are due to unforeseen circumstances such as the year 2000 problem. Software has changed the nature of communication and the availability of information. Social networking sites allow us to share virtually any aspect of our lives, but the public nature of such information has lead to widespread concerns with regard to cyber safety and privacy.

The internet allows access to an amazing and almost unlimited store of information, but filtering and verifying the correctness of such a vast store of information remains an issue.

The Year 2000 (Y2K) Problem

In the 1960s and 1970s storage was expensive so software was required to use the absolute minimum amount of storage space. To cater to this need, software developers often used only 2 digits to represent the year within dates. It was never envisaged that these products would still be in use when the Year 2000 came along. As a consequence of this short-sightedness it was anticipated that many computer-based systems would fail as the year changed from 1999 to 2000. That is, the software would potentially think that the year was 1900 and not 2000. Similar problems were expected with regard to other years, such as 1960 being interpreted as 2060 and hence causing a variety of incorrect calculations and computer crashes.



GROUP TASK Discussion
Many new companies emerged during the 1990s with the sole aim of checking software for Year 2000 compliance.
Should consumers and businesses have had to utilise the services and products of these 'Y2K' companies?
Discuss and debate each side.



GROUP TASK Debate
Many individuals and social groups prepared for the worst disaster. They felt that communication systems, power and other utility supplies would fail as a result of the 'Y2K' problem.
Discuss why these people may have had grave concerns.
Do you think that these concerns were justified?



GROUP TASK Discussion
As we now know there were no significant system failures as a result of the 'Y2K' problem. Let's for a moment consider who would have been responsible had there been a complete failure of the electricity grid system on a global scale.
Discuss in terms of social and the ethical considerations.

Computer Malware (including viruses)



Malware
Malicious software designed to damage or perform unwanted actions on computer systems.

Malware or malicious software includes any software which performs an unwanted and uninvited task. Examples of malware include viruses, worms, Trojan horses and spyware. New malicious software appears somewhere in the world on a daily basis. Anti-virus, anti-malware and anti-spyware utilities are installed to detect and destroy malware and to advise users of potentially unsafe practices or websites.

The ability to replicate itself distinguishes viruses from other malware. Viruses are commonly attached or embedded within apparently legitimate software or files. As the software is executed the virus is also able to execute, duplicate itself and perform its malicious processing. The duplicates are then passed onto other computer systems via email and other forms of electronic communication.

Worms are a type of virus which aims to slow down a system. Many worms exploit a security flaw in the operating system and then install themselves on the hard drive (often as hidden files). The worm then begins to replicate itself over and over again. Over time the worm consumes the entire hard disk. Other worms continue to execute copies of themselves. Typical symptoms of such attacks result in RAM being completely consumed by the worm and hence other software will run slowly or there may be insufficient RAM for software to execute at all.

In Greek mythology, a Trojan horse was a large wooden horse which the Greeks offered as a sign of peace to the city of Troy. Once the Trojan horse entered the city gates the Greek soldiers jumped out and apparently overthrew the city. In computer terms a Trojan horse is malicious software that is masquerading as legitimate software. For instance, you may think you're downloading a game, however when the game is executed it begins to perform unwanted processes, such as deleting or corrupting files.

Trojan horses are often used to install malware that enables an outside user (or program) to take control of your computer. Such Trojan horses are used to create botnets (or zombie armies). Sometimes these botnets include thousands or hundreds of thousands of computers. The owner of the botnet is able to instruct all the computers to perform malicious tasks. For example, denial of service (DoS) attacks occur when all computers in a botnet are instructed to access a single website. The web server is unable to process all the requests and hence the website crashes or is unable to process legitimate requests. Such attacks can come from competitors who wish to cripple the competition. Botnets are routinely used to send spam messages. Often all computers within the botnet are instructed to send the same email to all contacts. The botnet provides a level of cover for the criminal operator as it appears the spam messages are from a legitimate source.

Spyware aims to obtain your information without your knowledge. Some spyware is able to capture your web browsing habits, read your email contacts or even read keystrokes as you enter passwords and credit card details. The information is sent back to the criminal who then fraudulently uses the information for various illegal acts. The information obtained by spyware assists criminals to steal your identity. Once they have sufficient personal information the criminal is able to pretend they are you and gain access to bank accounts or perhaps take out a loan or credit card in your name. Identity theft has become an enormous issue not just because of the money being stolen, but also because it taints an individual's reputation. It can then be difficult for victims of identity theft to obtain credit.



GROUP TASK Discussion
Often free software is bundled into installation packages that contain adware. Adware causes annoying advertisements often as banners to appear. Do you think adware is malware? Discuss.



GROUP TASK Research
Each class member is to research a different specific example of malware. Determine how the malware is distributed, what malicious processes it performs and methods for detecting and removing the malware. Share your results with the class.



GROUP TASK Discussion
A company purchases a new software application and implements it across its network. A virus penetrates the application and corrupts all the company's data.
Who do you think is ethically responsible for the damage caused – the application developer, management of the company, or the employees?

Reliance on Software

We are often unaware of the impact of computers and software on our everyday lives. Household appliances such as ovens, televisions, videos, washing machines and stereos all rely on software to operate. All types of motorised transport, from cars, buses and trains to aircraft, are controlled by software. Utilities such as electricity, water and gas also rely on software. Government authorities such as the Tax Office and Social Security as well as hospitals, banks, universities and other large institutions are all software reliant in some form or another. The software development industry has a huge responsibility to ensure all these systems are reliable and perform their various functions accurately.



GROUP TASK Discussion
Compile a list of items around your school environment that have the potential to cause problems for students and teachers should the controlling software fail.



GROUP TASK Discussion
An aircraft manufacturer has developed a computer system that automatically compensates, if an engine failure occurs, by increasing boost to the other engine. This system also adjusts the control inputs so that the aircraft's handling attributes remain almost identical to normal. Pilots were not informed of this system on the new aircraft.
Identify scenarios where this new system could actually be counter-productive to the intended purpose and potentially cause accidents.



GROUP TASK Discussion
How often have you heard someone say, "Oh, sorry, we cannot do…., the computer is down"?
Describe occurrences from your experiences when this has happened.
Are these occurrences the fault of the computer systems?
Why are the computer systems blamed?
How could problems like the ones you have described be avoided?

Social Networking

A social network is simply a group of people who associate with each other. Often we have a variety of social networks based on some common association. For instance we have social networks built around family, another for friends, one for work and maybe some for special interests such as sports or hobbies.



Social Networking Site
An online, virtual community where people share their thoughts and activities with family, friends or those with similar interests.

Recently the concept of social networks has rapidly expanded into the online world. Perhaps the most well known social networking sites are Facebook, LinkedIn and Twitter, but there are numerous others. The first recognised social networking site was SixDegrees.com, which operated from 1997 to 2001 and was based on the idea that there are a maximum of six degrees of separation between any two people on the planet. In 2002 Friendster and then just a year later in 2003 MySpace commenced operation. MySpace was created by two friends and was the most visited website during 2006. It was sold to Murdoch's News Corporation in 2005 just two years after its launch for a reported $580 million. However by 2008 Facebook (which commenced in 2004) had overtaken MySpace as the top site.

Facebook was developed by a group of college students in 2004. Initially it was only open to students, but it was when it was released to the world that it really took off. The movie "The Social Network" released in 2010, describes how Facebook was founded and exponentially grew to become the most popular social site of all time.

Twitter, which has its own unique instant messaging approach, commenced in 2006. Essentially short messages (called "tweets"), including SMS messages from mobile devices, are added to live feeds. Subscribers can view these feeds and also group "tweets" based on hashtags. For example, the hashtag #sddhsc might be used to group "tweets" from current SDD students.

LinkedIn is a business contacts networking site. Professionals make business contacts and are able to share these contacts with others. Commonly a chain of contacts evolves yet the various links between ends of the chain remain private. For instance a professional contact can be made without knowing any of the people that form the links between the two ends of the chain. Currently social networking sites are becoming increasingly more integrated, for instance twitter feeds can be included on Facebook and LinkedIn pages and vice versa.

Clearly social networking sites have many varied and worthwhile advantages, however there are also disadvantages. Private and personal information is all too easily shared and this can (and has) led to a variety of problems including identity theft, stalking and online bullying. The ability of a single message to be delivered to thousands of unknown others has the potential for unintended consequences, such as thousands of people arriving at a party. Comments made on an online site can remain accessible indefinitely. Views you may hold now and comments made as a teenager may be embarrassing and inappropriate when viewed by some future employer or by your future children and friends.



GROUP TASK Discussion
Brainstorm a list of disadvantages of social networking sites. Discuss strategies that could be used by users of these sites and also by the developers of these sites to minimise the effect of the disadvantages you have identified.



GROUP TASK Research
Most social networking sites utilise a database to store all user data. Software then reads from the database and formats the information into a webpage for delivery to the user. Research a particular social networking site to determine the database system used and the programming languages used.

Cyber Safety

The online world of the internet includes a variety of different dangers and security issues. All internet connected devices have risks, this includes laptops, mobile phones and even game consoles. Cyber safety is about minimising the risk of such dangers, particularly for children. Although security software and firewalls can assist in this regard, many online issues are unable to be dealt with using software alone. Cyber safety is about protecting yourself when online. To be cyber safe means observing recommended practices and procedures to protect your personal information when using the internet.

The Australian federal government has setup the Cybersmart website (www.cybersmart.gov.au) to inform users about cyber safe practices and procedures. The information below is a brief summary of some of the recommendations from the Cybersmart website:

- Location based services. Services such as Google latitude and Facebook places, allow your current location to be tracked. This makes it simple for unknown persons to find you and it also alerts them that you are not at home. It is critical to check privacy settings regularly to ensure only those you know and trust are able to view your location. It is probably wise to turn off location tracking unless it is specifically required.
- Unwanted contact. Whilst online you may receive messages or see something that makes you uncomfortable. Do not respond to such messages; rather tell someone you trust, such as a parent or teacher. It can be useful to retain a copy of any unwanted contacts in the event that it continues as such evidence can greatly assist when tracking down the perpetrator.
- Cyberbullying. When online the same standards apply as in real life. Spreading false rumours, teasing or making threats online is unacceptable. The best defence is to ignore or block the cyberbully. Often a friend will be the victim of such attacks. It is best to support them and do not forward such messages to others as this will only exacerbate the problem. Instances of cyberbullying should be reported to the appropriate authority - your school or the police.
- Online friends. Meeting new friends online is fun and a great way to discuss common interests. Be aware that people may not be who they claim to be. It is prudent to not share private information to those you do not know in real life. Try to manage your friends list wisely. Having hundreds of online friends can be exciting, but it is very difficult to keep track of such a wide and varied group. If you have not had contact with an online friend for some time it is best to remove them from your contact list. Should you wish to meet an online friend in person never go alone. Take a trusted adult along, such as a parent, and always arrange to meet in a public place.
- Your digital footprint. What you share with friends is often made public. For instance, a photo or video you post for a friend can easily be forwarded to others who then post it publicly. Furthermore these messages and private information can remain accessible forever. When all your online information is combined a picture of you emerges - your digital footprint. Will this digital footprint cause issues later in life? For example, if read by a future employer, partner or even your future children. It is virtually impossible to remove such information completely. Also, respect your friend’s privacy and think about the possible consequences before forwarding their information to others.

- Online purchasing. It is prudent to restrict online purchases to well known organisations, such as eBay or brands that have real shops. Examine the details and conditions for sale. Be wary of ongoing purchases where your credit card will be debited many times. Most trustworthy sites will include a system for contacting sellers should you be unsure about the goods, postage or returns policy. Never enter credit card or bank account details without ensuring the transaction is securely encrypted.
- Identity theft. Criminals can obtain sufficient information about you so they are able to use your credit card, access your bank account or take out loans in your name. Be very careful with your personal information. Use virus and/or malware utilities to protect against spyware. Ensure you use complex passwords and protect these passwords. Do not respond to emails that request passwords or any other financial information. And finally, watch your bank accounts for any unusual or unauthorised transactions. Often a small withdrawal of a few cents indicates a criminal testing your bank account before they make much larger withdrawals.



GROUP TASK Research
Research an example of cyberbullying. Outline the nature of the bullying, the consequences for the victim and explain how others could have assisted to prevent the bullying occurring in the first place.



GROUP TASK Discussion
Currently security software and firewalls have limited value in terms of cyber safety. Explain reasons why software solutions are unable to effectively keep us "Cyber Safe"?

Evaluating information available through the internet

Huge amounts of information are publicly available through the internet. Information is uploaded or posted by anybody regardless of their qualifications, expertise or experience and in many cases it is difficult to identify the author. This rarely occurs with published print and broadcast media where the author is clearly identified and all material is checked by a variety of skilled personnel, such as reviewers, editors and proof readers. On the internet biased, unsupported, unverifiable, misleading and often incorrect information is common. Although the internet encourages and enhances freedom of speech for all, it has the unfortunate consequence that determining the correctness of such information is difficult and often impossible. This is a particular issue when performing research as verifying the accuracy of information is a necessity.

The following questions and guidelines can assist when evaluating all information sources, but they are particularly important when the information is sourced through the internet. Unfortunately it is often difficult to obtain definitive answers to these questions for much of the information available on the internet, therefore we often need to uncover evidence and then do a little bit of detective work to make an accurate judgement.

- Who is the author? The author's name should be clearly stated. If it is a person (rather than an organisation) then look for their qualifications and or claimed expertise in the area. Often information on an educational or government institution's website has more authority. Phone numbers and street addresses also

improve confidence in the legitimacy of the author. Try to search for other documents which reference the author to confirm their expertise and other details. Many professionals will maintain their own web pages; such pages can assist in verifying their identity and authority.

- Is the information up-to-date? Look for a date on the page or examine the HTML source for a date. It is better if the page clearly states when and how often it is updated. Depending on the topic, it is often obvious if the information is reasonably up-to-date. For example, it may reference some event that has occurred recently.
- Who is the intended audience? Is the information written for the general public or is it written for experts in the area. It can be difficult to determine the validity of information for the general public, whilst expert information is likely to include a list of references.
- Is the information accurate and unbiased? Is the information supposed to be a factual account of the topic or is it the author's opinion. Be wary of information of a political, religious or cultural nature as it can be biased to a particular view of the world. Commercial sites are also likely to be understandably biased towards their own products and services.
- What is the purpose of the information? Be careful of information whose purpose is to market a product or persuade you to agree with their point of view. The purpose of the information should be clear and the information should achieve this purpose. Some commercial sites may claim to be educational, when in fact the purpose is to advertise and promote.



GROUP TASK Research
Wikipedia is often the first website students visit as they perform internet research. Research and explain how information is obtained for inclusion on Wikipedia and how the accuracy of the information is verified.



GROUP TASK Discussion
Books and published articles in magazine and journals are considered to be more authoritative sources compared to most internet sites. Do you agree that this is generally true? Discuss.

RIGHTS AND RESPONSIBILITIES OF SOFTWARE DEVELOPERS

Software developers have *rights* in regard to how other people may use and duplicate their products. Along with these rights come *responsibilities*. Developers of software products are accountable to their customers to provide solutions that are of high quality, ergonomic, inclusive, free of malware and above all perform their stated task correctly and efficiently. Responsible developers will provide systems that allow them to respond to problems encountered by users. Many developers voluntarily adhere to a professional code of conduct.

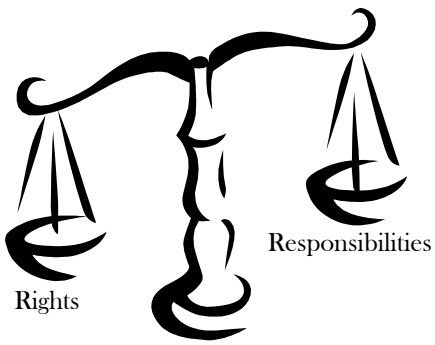


Fig 1.1
A balance between the rights of developers and their responsibilities to customers must be found.

INTELLECTUAL PROPERTY



Intellectual Property
Property resulting from the fruits of mental labour.

Intellectual property is property resulting from mental labour. Therefore intellectual property is created by all types of authors, including software developers. An author is the *original creator* of a work, and hence they own the right to control how the work is sold and distributed. With most literary works, such as books, poems and essays it is clear who the author is and therefore it is clear who owns the intellectual property rights. In the case of software products, the situation is often not so clear.



Consider the following:

- A software product is written in a programming language, which was developed by another company.
- A team of developers is utilised to create the software product, with some of those developers being outsourced from another company.
- Some modules of source code are purchased from other software companies.
- The printed documentation is designed and written by a company, which specialises in computer user manuals.
- Finally, the artwork for images, video and animation is designed and created by a graphic design company.

All the people and companies involved in the original creation of this software package are authors and have intellectual property rights in regard to their particular original contribution. *Copyright laws*, discussed later in this chapter, are designed to protect the intellectual property rights of all the authors involved.

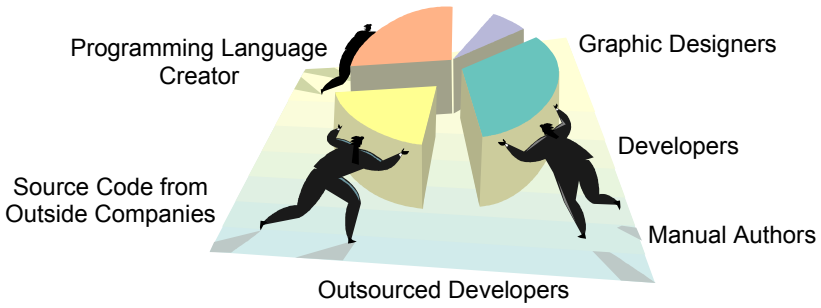


Fig 1.2
All authors have intellectual property rights in regard to their original material.



GROUP TASK Discussion
Software developers expect their intellectual property rights will be recognised. They must also respect the intellectual property rights of others who have contributed to the software they produce. Discuss.



GROUP TASK Discussion
Do you think each of the contributors to a software product should have intellectual property rights? What if the author is employed by a software development company? Discuss.



Royalty
Compensation, usually a portion of proceeds paid to the owner (author) of a right, for the use of a work.

Different contributors to a software product will have differing requirements in regard to their intellectual property. Most programming languages are either open source or they are sold with a licence requiring the developers to acknowledge use of the programming language without the need for royalties to be forwarded. Original work, written for a specific product, usually requires a royalty to be paid to each contributing author based on the number of units sold of the final product. The particular details of how royalties are to be distributed should be detailed in a contract.



Publisher
One whose business is the copying, distribution and sale of works by an author.

Publishers of software products have a responsibility to ensure the intellectual property rights of all authors involved in the creation of the final product are acknowledged and compensated, as the law requires. As was seen previously, this process often involves many individuals and companies.

QUALITY

The final quality of a software development project is an important responsibility for all software developers. To develop high quality applications is time consuming and costly. Often compromises have to be made because of financial constraints.

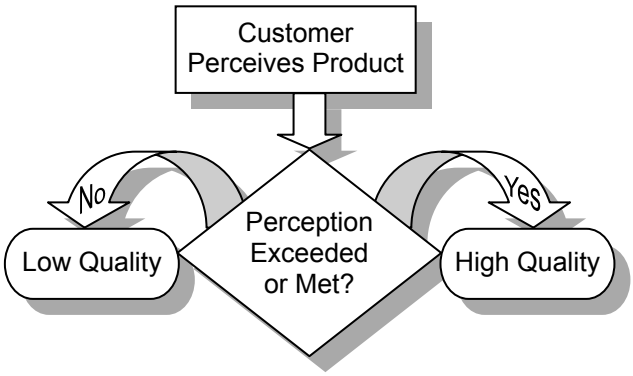


Fig 1.3
Customer Perception of Quality.

Quality software is developed as a result of thorough planning and testing. Many companies have developed sets of standards to which their employees must adhere. *Quality Assurance (QA)* is used to ensure that these standards are observed. It is an ongoing process throughout the design and development to ensure and verify the quality of the software. Customers perceive quality according to their expectations about the software product. If the customer's expectations are exceeded, then they will perceive the product to be one of high quality. If the product fails the customer's expectation then the product is perceived to be of low quality.

Quality Assurance attempts to make sure that customer expectations are met and/or exceeded. Different customers will come with differing expectations. However, all customers have a right to have their expectations met. It is the responsibility of the software developer to make sure this occurs. It is in the developer's best interest to know the customer's requirements and to exceed their expectations if they are to continue to operate profitably.

As with most industries and products, quality at a reasonable price is a formula for success. Software developers must strive throughout the software development process to produce high quality software products which exceed their prospective customers' expectations.



Consider the following list of factors that affect software quality:

- Correctness: Does it do what it is supposed to do?
- Reliability: Does it do it all of the time?
- Efficiency: Does it do it the best way possible?
- Integrity: Is it secure?
- Useability: Is it designed for the end user?
- Maintainability: Can it be understood?
- Flexibility: Can it be modified?
- Testability: Can it be tested?
- Portability: Will it work with other hardware?
- Re-useability: Can parts of it be used in another project?
- Interoperability: Will it work with other software?



GROUP TASK Discussion
Consider the above list of factors affecting software quality. Provide an example situation where each factor is not present and hence the quality of the software is compromised.

External factors affecting quality

Apart from the design and development process there are other external factors, which need to be considered by developers to maximise the quality and reliability of their products. Such factors include:

- Hardware
- Operating System
- Other software
- Runtime errors

- **Hardware**

The vast array of hardware devices, which includes various network devices, available today makes the job of the software developer difficult. Software developers rely on hardware manufacturers to design and build components that operate reliably on the given platform. The software developer has a right to expect this to be the case. To ensure maximum reliability of their product, software application developers should test their product with a number of different hardware configurations using a variety of different devices. Developers have a responsibility to ensure this testing is as thorough as costs and time allow.



Fig 1.4
A Computer Lab is often used in the testing of software.

It is inevitable that hardware problems will occur and it is the responsibility of the software developer to build software modules, which react appropriately to these hardware failures. Software, which helps to identify the cause of a hardware failure, or reports the actual failure, will prove to be more reliable and hence higher quality.



Consider the following:

A database for a large business will grow in size over time. It is inevitable that at some time the amount of secondary storage on the hard drive will be insufficient to store the database. The software developer should take this into consideration when designing the system so that warning messages are given when low levels of hard disk storage space occur. This will allow the user to either purchase more hard disk space, or to free up space on the current drive by way of purging old data, which is no longer required.



GROUP TASK Discussion
Should it be the responsibility of the software application to report that the low storage threshold has been reached, or should it be the responsibility of the operating system developers? Discuss.

- **Operating System**

The developer needs to consider the effects that the operating system will have on their software product. Developers have a responsibility to ensure as far as possible that their products interface correctly with the specifications of the intended operating system environments. At the same time, the operating system creators have a responsibility to ensure that specifications published are accurate and that the operating system performs correctly. Developers of software expect the operating system to be reliable and robust. Application software *should not* make changes to the operating system settings. Any changes made to the operating system that are not part of the original specification may cause the operating system to return unexpected results. Application software developers should instead implement checks in their product to ensure that it performs reliably given most operating system settings.

For example, some operating systems allow for the length and format of file names to be altered. Rather than the application changing the operating system settings to suit, the application should access the allowable format of file names and adjust in line with the operating system settings. This will ensure the reliability of not only the operating system but also other installed software applications.



Operating System
Provides the interface between the software application and the hardware.



GROUP TASK Discussion
Currently the developer's of operating systems have much control over the look and functionality of application software. What responsibilities do operating system developers have to application software developers? Discuss.

Operating systems are continually being improved. It is the responsibility of the software developer to ensure that their product is as *forward compatible* as possible. Conversely, it is the developer's right to expect that the developers of the operating system will continue to support products written to interface with earlier operating system versions in their upgraded products.



Forward Compatible
The ability of an older system to operate in harmony with a new system.



GROUP TASK Discussion
Most operating systems are regularly updated to correct bugs and security vulnerabilities. Many software applications are now providing a similar facility. Commonly these updates occur automatically via the internet, however it can be annoying for users who are continually bombarded with update notifications. Discuss in terms of developers and also users.

- **Other Installed Software**

Software already installed, or that could be installed in the future, can affect the perceived quality of a developer's product. Software developers have a right to expect that other developed products will not affect the operation of their product. Unfortunately, this cannot be relied upon. Therefore, the developer has a responsibility to ensure that as far as possible other installed software is not able to affect the reliability of their product. Of course, software developers must be careful to ensure that their product also does not adversely affect the operation of other software products.



Consider the following:

A fax program that does not correctly close the serial port connected to the modem will cause other communication programs to encounter errors when accessing the modem. By predicting this as a possible occurrence, the developer should write a module that checks the serial port status before attempting to access the modem. The software can then prompt the user to close the port before attempting to open it and use the modem.



Fig 1.5
Multiple communication programs may access the modem.



GROUP TASK Discussion
If all software developers did the right thing and cleaned up after their applications closed, then problems such as this would not occur. Perhaps standards are required to ensure software developers perform these 'clean-up' tasks. Do you agree? Discuss.



GROUP TASK Discussion
Identify other possible scenarios where installed software applications may cause conflicts with the operation of other applications.

Many problems that jeopardise the integrity of a software product's ability to perform in a reliable manner in conjunction with other installed applications occur at installation time. Shared files can be overwritten without considering the effect on other installed applications. Often file associations are altered at installation time without warning. These actions result in reliability problems for other developers that are difficult to overcome. It is the responsibility of software developers to consider the rights of other software products when coding the installation procedure of their product.



File Association
Defines the application that is executed when a data file is opened, usually by double-clicking.

- **Run Time Errors**



Shared File
A file whose code is used by more than one program. Example: A dynamic link library file (.dll file)

All applications developed must include error checking built into the code. Error checking enhances reliability and minimises *run time errors*. It is arrogant to presume that a program will be completely error free, and in any case, it is impractical to test for all possible eventualities. However, to minimise the possibility of errors, thereby ensuring greater quality, the developer must include routines that will deal with possible errors in execution. It is the software developer's responsibility to ensure that errors encountered during the execution of the code are dealt with in such a way that execution continues and the product can continue to execute. Fatal errors, which result in execution being terminated, can result in loss of user data. Therefore a developer will find it advantageous, when writing a software application, to code a fatal error detection module which executes a save procedure prior to execution being terminated.

RESPONSE TO PROBLEMS

Software developers have a responsibility to ensure that any problems users encounter with their products are resolved in a timely, accurate and efficient manner. The developer needs to ensure there is a mechanism in place to assist in the identification of errors and their subsequent resolution. Error messages issued by the application should be unambiguous and should aim to identify the source of the error.

Most large software companies have support departments whose task is to respond to user problems. Even small developers must ensure that they have systems in place to guarantee a quick response to problems encountered with their products. Many support contracts specify a maximum time allowable for a problem to be rectified. Often businesses outsource their customer support to a company that specialises in computer support.

Depending on the severity and how often the error has occurred, different methods of response will be more appropriate.

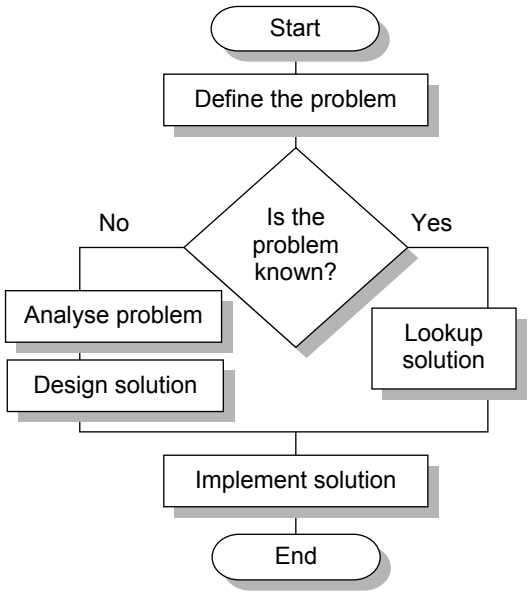


Fig 1.6
The processes undertaken to solve a user problem.

Problems resulting in the product being unusable must be resolved with the utmost urgency. Urgent attention must also be given to problems encountered by numerous users. Less significant problems may be ranked according to their importance and resolved as time constraints allow. Many software support companies allocate a severity/priority classification to each problem reported. Examples of these are as follows:

1	CRITICAL	Affecting operations/financial loss	Immediate Resolution
2	MAJOR	Affecting operations – difficult workaround available	Resolution up to 3 months
3	NORMAL	Affecting operations – workaround available	Resolution up to 12 months
4	COSMETIC	No effect on operations/Enhancement requests	As time constraints allow

Fig 1.7
Priority/Severity table.



Consider the following:

A software application is used by builders that assist in the allocation of resources to particular building projects. The software does not currently allow for tax legislation changes, which are to be introduced next financial year. Another problem emerges with this software that under certain circumstances causes delivery dates for materials to be delayed by one day. This problem will have drastic results, as tradesmen will arrive on site before their materials.



GROUP TASK Discussion
Both of the problems identified in the building software application need resolution. Which category from the *Fig 1.7* Priority/Severity table would each of these problems fit? Discuss your reasoning for allocating each category.

CODE OF CONDUCT

A Code of Conduct is a set of standards by which Software Developers agree to abide. Many professional software developers are members of professional associations that expect adherence to a written Code of Conduct. It is the responsibility of the individual developer to uphold the Code of Conduct. Doing so enhances the reputation of themselves as well as the profession as a whole. In the long term, accepted Codes of Conduct increase standards across the entire industry. Most professional associations have rules, which allow for members to be expelled if they fail to uphold the association's Code of Conduct. Clients of members of the association can complain to the association if they feel one of its members is not transacting business in accordance with the Code of Conduct. Members of the association should encourage adherence to the Code of Conduct by other members of the association.

Code of Conduct: Professional

1. Strive to achieve the highest quality, effectiveness and dignity in both the process and products of professional work.
2. Acquire and maintain professional competence.
3. Know and respect existing laws pertaining to professional work.
4. Accept and provide appropriate professional review.
5. Give comprehensive and thorough evaluations of computer systems and their impacts, including analysis of possible risks.
6. Honour contracts, agreements and assigned responsibilities.
7. Improve public understanding of computing and its consequences.
8. Access computing and communication resources only when authorised to do so.

Code of Conduct: Leadership

1. Articulate social responsibilities of members of an organisational unit and encourage full acceptance of those responsibilities.
2. Manage personnel and resources to design and build information systems that enhance the quality of working life.
3. Acknowledge and support proper and authorised uses of an organisation's computing and communication resources.
4. Ensure that users and those who will be affected by a system have their needs clearly articulated during the assessment and design of requirements; later the system must be evaluated to meet the requirements.
5. Articulate and support policies protecting the dignity of users and others affected by a computing system.
6. Create opportunities for members of the organisation to learn principles and limitations of computer systems.

Fig 1.8
A typical professional association Code of Conduct.



GROUP TASK Discussion
How can a 'Code of Conduct' affect the quality and reliability of software products? Why would software developers choose to follow a 'Code of Conduct'? Discuss.



GROUP TASK Activity
Develop a 'Code of Conduct' for use by the class during the development of your software projects. Explain how adherence to this 'Code of Conduct' could be monitored.

MALWARE

Software developers have a responsibility to ensure their products do not contain malware. To make sure this is the case they should check any new data and software being added to their computers for viruses. Policies should be in place to which all employees must adhere to make sure this occurs.

- All incoming emails scanned for viruses and other malware.
- Removable media scanned for malware before use.
- Update anti-virus software regularly.
- Scan all software before distribution to purchasers.
- Use firewalls and other security mechanisms to minimise the risk of security breaches.

Purchasers of software have the right to expect software they purchase is free of all malware. Software developers can be held responsible for distributing malware with their products.

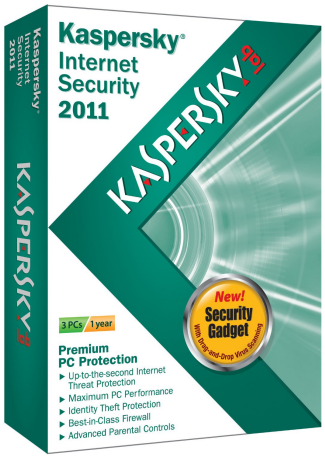


Fig 1.9
Commercial security software including anti-malware, anti-virus, anti-spyware and other security functions.

ERGONOMIC ISSUES IN SOFTWARE DESIGN



Ergonomics
The study of the relationship between human workers and their work environment.

Ergonomics is the study of the interactions between human workers and their work environment. In Software Design and Development we are concerned with the ergonomics of software solutions. As the user interface provides the connection between software and users the design and operation of the screens is of primary interest. Determining user needs with regard to their work routine and preferences greatly assists in the initial user interface designs. Usability testing should then be performed throughout the design process to ensure the software is ergonomically sound. User support, including help screens and other forms of user training and assistance, is another area of ergonomic software design that requires consideration.

User needs from an ergonomic perspective

In chapter 3 during the identification of the problem we shall consider user needs from a functional and performance perspective. This aims to ensure the code will perform the desired tasks correctly and in a timely manner. However, software that performs all functions correctly is of little use if users are unable to operate the software with ease. We should therefore analyse not only needs in terms of what the software should do, but also how users would prefer to do it. The software must make sense to the user and should operate in a manner that integrates with their way of doing things. Identifying such usability needs is not a trivial matter, particularly when there is no existing system from which to draw experience. Task analysis is often undertaken whereby the user is observed performing different tasks and their actions are noted. It is often useful to record other software applications that are routinely used so that consistency of design across applications can be assessed.

User interface design



User Interface
The screen designs and connections between screens that allow the user to communicate with software.

An intuitive user interface that allows users to work the way they want is perhaps the most critical aspect of successful software products. Such products are easy to learn and use. Fantastic functionality is of little value if users are unable to understand and operate the product. On larger software projects specialist user interface engineers are employed to ensure high standards of usability are maintained.

The most important aspect of user interfaces is consistency of design. Consistency of design allows users to transfer skills from other applications and other areas of your application to new areas. A mental model of the software application's operation is built up in the user's mind; consistency reinforces this model.

Some points vital to the development of consistent user interfaces include:

- Setting standards and sticking to them. Set design standards based on industry standards and then develop extra standards specific to your application and user's needs. Most operating system developers have sets of design standards; these standards will often define much of the design.
- Explain the rules. Develop a simple set of rules that apply to your entire application. In this way, you will need to explain your rules only once. This makes more sense than explaining the details of how to use each individual feature of your application.

- Use interface elements correctly. Know when to use screen elements and how to use them correctly. For example, command buttons, check boxes, radio buttons, list boxes, text boxes, menus, etc. Follow recognized standards for each of these elements. Never alter the operation of standard screen elements to perform unexpected functions.
- Use colour appropriately. Colour should be used sparingly. Colour is viewed differently by each individual, what looks pleasing to you may be abhorrent to others. Don't override a user's system settings, they chose them and your application should not override their choices. This ensures consistency between applications. Ensure you maintain sufficient contrast between text and backgrounds.
- Use fonts appropriately. The use of fancy fonts can decrease usability considerably. Use fonts that are easy to read. Don't override system settings for standard fonts. More than three fonts on a single screen are too much. Use fonts consistently, there should be a reason for font changes and the reason should be immediately and consistently clear. Remember you are using a different font every time you change the size, style, typeface or colour.
- Alignment of data entry elements. Text boxes should be left justified with their associated labels also left justified (see *Fig 1.10*). For columns of data, right justify integer, align decimals to the decimal points and left justify text.
- Provide a consistent method of reversing actions. Users like to explore software applications and are therefore likely to initiate processes that they later wish to reverse. Many software products include an undo function which reverses previous actions. Others require the user to confirm operations that will make extensive changes to data.

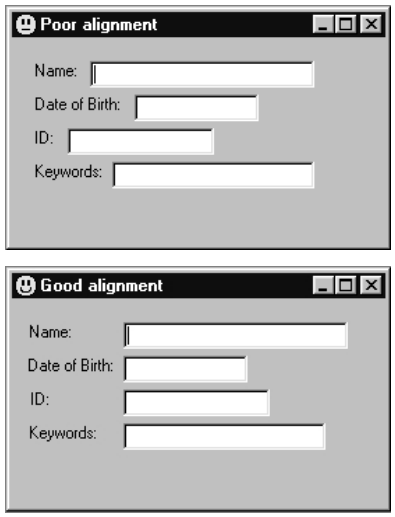


Fig 1.10
Alignment of labels and text boxes.

Usability testing

Usability is a measure of how successfully your software product meets the needs of your users. It is not so much about the performance and functionality of the product, rather it is about the efficiency and satisfaction users experience as they use the software. This is not easy to achieve and hence usability testing needs to be an ongoing priority throughout the development process. A variety of testing mechanisms can be utilised. For instance, prototypes of each screen can be distributed to specific users for comment, representative users can be included on the development team (which is integral to the agile approach) and task analysis can be undertaken with random groups of users. Whichever techniques are chosen the testing should be repeated once the identified usability issues have been corrected. That is, the best usability testing is iterative. It progressively refines the user experience until an optimal solution is reached (or time and budget is exhausted).



GROUP TASK Activity
Locate a software application on the internet which nobody in the class has used, but which performs some familiar task (such as resizing images). Use the software to perform the task. Discuss usability issues encountered by different students and recommend improvements to the design.

INCLUSIVITY ISSUES

Inclusive software should take into account the different users who will likely use the product; as software developers, we have a responsibility to ensure software is accessible to all regardless of their culture, economics, gender or disability. Furthermore, software products that do not take into account the different characteristics of users are less likely to secure a significant market share, which would result in lower sales and reduced profits.



Inclusivity
Not excluding people based on their cultural background, economic background, gender, disability, etc.

Cultural background

The culture of a people can be described as the set of ways of living built up over a period of time and passed from generation to generation. It is important that the beliefs and language of different cultures be considered when designing software. Similarly, the social structure of societies is influenced by their underlying culture.

How do we cater for these differences in practice? Firstly, we must understand or at least be empathetic to the needs of other cultures. It is not possible to be an expert on all cultures; however we can easily include users from a variety of cultures as part of the testing processes occurring during development. For example, in most western cultures we have a Christian name and a surname. This is not the case in many Asian cultures where a formal name and an informal name are more commonly used. Testing that includes users from the Asian culture, would quickly highlight this difference.

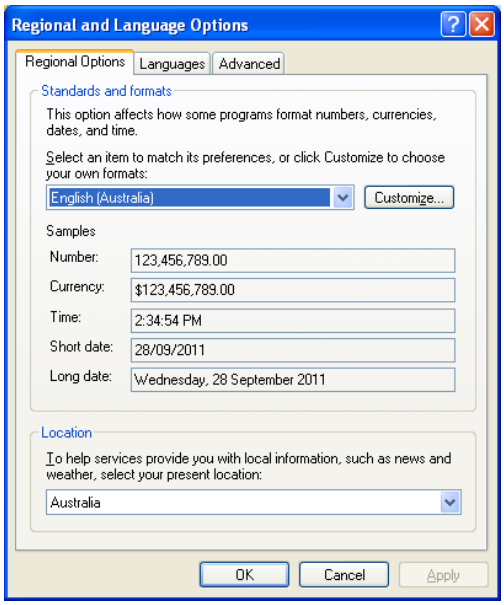


Fig 1.11
Windows allows the format of numbers, currency, times and dates to be altered.

Numbers, currency, times and dates are another common area of difference. In Australia, we express dates using day then month then year (e.g. 25/1/2012). In America, they more commonly use month then day then year (e.g. 1/25/2012). Again, those used to an alternative format, would soon encounter problems. Similarly, the format of numbers and currency is different in other countries. Sweden uses a comma as its decimal point and a period as a multiplication sign. 5.2 in Sweden would mean 5 times 2 and 3,255 would mean 3 point 255.

The dominant language of a country should be used on the user interface when applications are to be sold to those cultures. Many large-scale systems manage to utilise any number of languages by storing the text used to label the interface in a text or XML file that is loaded as the application starts. By altering this single file the language used by the application can be customised to suit any foreign language. It may not be practical to rewrite our interfaces using a large variety of foreign languages, however we can design them using English that is clear, consistent and unambiguous. In this way users whose first language is not English are more able to comprehend our user interfaces.

Cultural differences are often prevalent in regard to religious beliefs. Many religions worship at a particular time during the week. Perhaps software can be written so that processor intensive batch operations can be performed at this time. Attempts at humour within software can often be in bad taste to those with different religious beliefs. Similarly graphics, particularly many appropriate in western cultures, are deemed inappropriate in many eastern cultures. Increased awareness of the audience, including minority cultures and religions, is a vital step in developing culturally inclusive software.



GROUP TASK Investigation
Examine a number of programming languages to determine features which allow source code to format dates, times and currency according to the user's location.

Economic background

Economic characteristics relate to the generation, distribution and use of income and wealth. This can be viewed on a global, national, local or industry specific scale. Software development is a relatively new industry that is growing in both breadth and overall volume; a very healthy economic situation. As software developers, we have a responsibility to ensure consideration is given to the economic situation of purchasers of software products. This is necessary to ensure the industry maintains a solid position in the market place in the years to come. Unlike most other industries, the cost of software products is most significantly influenced by its design and development costs; the production costs being relatively insignificant. To achieve equality of access to technologies such as software requires that the technology is available at a cost that is economically viable for the widest possible audience. The recent trend towards open source development helps in this regard. Larger commercial organisations access additional paid support and services whilst poorer organisations are still able to access the same quality software at minimal cost.

Let us consider some issues related to design and development costs, which in turn influence the final purchase price for users.

- **Quality**

Quality is a measure of how well a product meets the needs of its clients. A product that better meets the needs of the user will be more successful, however quality costs money. A software product developed for a single user cannot economically be produced to the same quality standards as one produced for the global market. A balance must be found between cost and quality.

- **Nature of the market**

Software developers must understand the market needs before embarking on new product development projects. It is not economically feasible to develop a product where the need it is intended to meet is of less value than the product's cost. A feasibility study should be undertaken to determine economic feasibility of any new software project. As software developers, we should acknowledge that some market needs are better met by non-computer based solutions.

- **Management of the software design and development process**

There are various software development approaches, (see chapter 2) suited to the creation of different products. Choosing a suitable approach will result in a more economically viable product. Different work environments can also influence those involved in software development. It is often possible for developers to work flexible

hours and to work from home. By creating structures that help developers work more effectively, managers will reduce costs and increase the quality of the software produced.

- **Influences on pricing**

Companies are ultimately in business to make money. This is the basis of most western capitalist economies. History shows us that companies that have a monopoly in their industry tend to produce inferior products at higher costs to consumers. As a result these companies make large profits. Competition and in some cases government regulation can assist in ensuring software products are sold at realistic prices and are within the financial grasp of a wide audience.



GROUP TASK Research
The Apache web server and MySQL database management system are examples of popular open source software. Both Apache and MySQL are available free of charge. Research to determine how these two products are able to remain economically viable.

Gender

In most cases the first thing we ask when a new baby is born is whether it is a boy or a girl. In actuality, this is probably the first thing we perceive when we meet anybody for the first time. We seem to inherently know and sense the differences between the sexes. We don't need to view a face or hear a voice; small almost imperceptible cues almost always allow us to correctly identify the gender of a person. So what are the major differences and how should these differences be included when designing and developing software?

Firstly, both men and women should be included in the software design and development process. At the time of writing, men dominate the industry. Many view this as a natural consequence resulting from one of the major perceived differences between men and women. Programming is viewed as a technical, mathematical process with rigid boundaries. Research shows that rightly or wrongly, men tend to dominate in these types of occupations. Since men are engaged in the creation of software, it follows that some bias is likely to exist towards males within the software products they develop.

Computers generate precise scientific-type responses; this is the nature and construction of the machines. In general, research has shown that men prefer black and white situations; they wish to reach a definite solution. On the other hand, women often see problems in a broader sense. They are interested in ensuring the method of solution is socially acceptable and are generally better at reaching and accepting more flexible solutions. Software should be designed to allow for both styles of solution. Traditionally software has been written to solve problems that do have definite answers (e.g. a set of inputs is processed into a unique set of outputs). Current advances in artificial intelligence now allow computers to assist in the solution of a much broader range of problems using less structured solution techniques.

Disability

Software design should include functionality that allows software to be used and accessed by a wide range of users. The computer and its associated software has revolutionised the lives of many disabled people. It is estimated that one in five people have some form of disability that will affect their capacity to access and use software in the usual way. To ensure maximum accessibility to those users with a

disability, software designers have a number of tools and techniques at their disposal. Let us examine some broad categories of disability and discuss some accessibility techniques that should be considered by software designers.

- **Visual disabilities**

Simply using larger fonts and providing the facility for controls and graphics to be scaled may overcome reduced visual ability. Most operating systems provide these features and software developers should ensure they utilise the operating system's settings within their products.

Colour blindness and other related conditions make it difficult to distinguish different colours and colour combinations. For this reason, colour should not be used as the sole method of conveying information. For example, internet links are often blue, however they should also be underlined.

To accommodate users with severe to total impairment requires software to correctly interface with speech generation and Braille utilities. These utilities are able to read the text labels on the user interface and convert them into speech or Braille. In this way, the utility is able to track and describe what the user is doing. Graphics can be a problem for these readers; it is recommended that labels be used to identify all functional graphics.

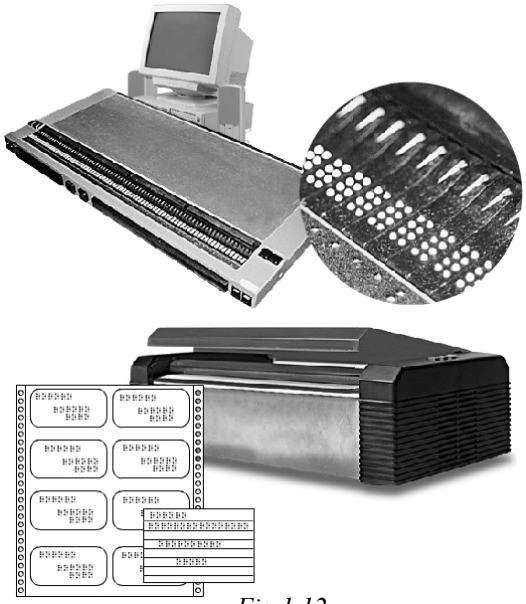


Fig 1.12
The PowerBraille (top) is both an input and output device. The BrailleBlazer (bottom) is a Braille embosser for creating Braille hardcopy.



GROUP TASK Research
Search the Internet for both software and hardware to support the visually impaired. Compile a list of devices and software found, together with a brief description of the purpose of each one.

- **Hearing disabilities**

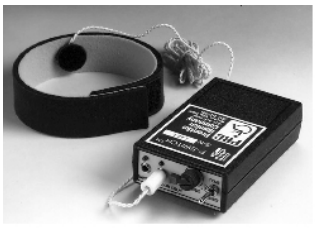
Software should not rely on sound as the sole method of communicating information. It should be used to emphasise and/or reinforce other visual clues. Utilities are available that will give some visual clue if a sound has been produced.

Software developers should be aware that for many hearing-impaired users English is their second language, their first language being sign language. There are a variety of software utilities available to convert speech into text or into sign language.

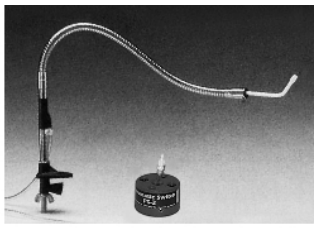
- **Physical disabilities**

Some users have difficulty performing certain physical tasks. They may not be capable of using a mouse or they may find it difficult to reliably press a single key on the keyboard. Other disabilities may severely limit physical movement to the face or even just the mouth. For these people, the ability to use computer software is vital if they are to successfully communicate. Many software utilities that interface with specialised input devices for the disabled utilise the standard keyboard interface; in effect they send the same data as a standard keyboard. It is therefore vital that the user interface is able to be controlled using just the keyboard.

Accessibility features found in many operating systems and applications to assist the physically disabled include: altering the time-delay before keys repeat when held down; allowing keyboard combinations to be entered one key at a time e.g. pressing Ctrl and then pressing S without the need to hold down Ctrl; changing the keyboard layout; simulating the mouse using the cursor keys; changing the mouse pointer speed and adjusting the double-click speed.



P-Switch controlled by fine muscle movements in the eyebrow area



Sip and puff switch controlled by the users breath



Large movement rocker switch controlled by the arm or chin



Tongue switch

Fig 1.13
Switches are used to provide the input to specific accessibility applications.



Fig 1.14
Windows XP onscreen keyboard interfaces with most applications and specialised input devices.

Specific applications are available to perform input and output functions from and to specialised hardware devices. Many of the input devices are simple switches of varying size and sensitivity to suit the needs of the individual user. For example, users without the use of their limbs are able to use head movement detection devices to control input to an onscreen keyboard. Various other examples of switches are shown in *Fig 1.13*. The software interfacing with these devices commonly uses short mnemonics and word prediction to speed up data-entry. Sentences that are commonly used can be pre-defined and completed with a single input.



GROUP TASK Activity

Windows, and no doubt other popular operating systems, include an onscreen keyboard (*Fig 1.14*) that can be configured for use with a single key (often the space bar). Type a sentence into a word processor using an onscreen keyboard and a single key. Comment on the difficulties encountered and suggest how these difficulties might be minimised.

PRIVACY ISSUES

Privacy is about protecting an individual's personal information. Personal information is any information that allows others to identify you. Privacy is a fundamental principle of our society, we have the right to know who holds our personal information. Privacy is a feeling of seclusion, where we can be safe from observation and intrusion. For this to occur we need to feel confident that our personal information will not be collected, disclosed or otherwise used without our knowledge or permission.

Personal information is required, quite legitimately by many organisations when carrying out their various functions. This creates a problem, how do we ensure this information is used only for its intended task and how do we know what these

intended tasks are? Laws are needed that require organisations to provide individuals with answers to these questions. In this way individuals can protect their privacy.

In Australia, privacy is legally protected under the *Privacy Act 1988* and its subsequent amendments. This act contains ten National Privacy Principles, that set standards that organisations are required to meet when dealing with personal information; the text in *Fig 1.15* briefly explains each of these principles.

What are the ten National Privacy Principles?

The following briefly explains what the NPPs mean for you.

NPP1: Collection - describes what an organisation should do when collecting your personal information.

NPP2: Use and Disclosure - outlines how organisations can use and disclose your personal information.

NPP3: Data Quality & NPP4: Data Security - set the standards that organisations must meet for the accuracy, currency, completeness and security of your personal information.

NPP5: Openness - requires organisations to be open about how they handle your personal information.

NPP6: Access & Correction - gives you a general right of access to your own personal information, and the right to have that information corrected, if it is inaccurate, incomplete or out of date.

NPP7: Identifiers - says that generally, Commonwealth government identifiers (such as the Medicare number or the Veterans Affairs number) can only be used for the purposes for which they were issued.

NPP8: Anonymity - where possible, requires organisations to provide the opportunity for you to interact with them without identifying yourself.

NPP9: Transborder Data Flows - outlines privacy protections that apply to the transfer of your personal information out of Australia.

NPP10: Sensitive Information - requires your consent when an organisation collects sensitive information about you such as health information, or information about your racial or ethnic background, or criminal record. Sensitive information is a subset of personal information and special protection applies to this information.

Fig 1.15
The ten 'National Privacy Principles' briefly described from the Office of the Federal Privacy Commissioner's website at http://www.privacy.gov.au

Consequences of the *Privacy Act 1988* mean that information systems that contain personal information must legally be able to:

- explain why personal information is being collected and how it will be used
- provide individuals with access to their records
- correct inaccurate information
- divulge details of other organisations that may be provided with information from the system
- describe to individuals the purpose of holding the information
- describe the information held and how it is managed



GROUP TASK Discussion
Many organisations legitimately store our personal information. Explain how software could assist in ensuring our personal information remains private and secure.

SET 1A

1. Compensation paid to an author is known as a:
   (A) right
   (B) royalty
   (C) commission
   (D) responsibility

2. Which of the following can be thought of as an author in regard to a software product?
   (A) Developer
   (B) Graphic designer
   (C) Programmer
   (D) All of the above

3. An organisation which copies, distributes and sells works is called a:
   (A) Publisher
   (B) Developer
   (C) Programmer
   (D) Retail outlet

4. In terms of ensuring reliability in their products, developers need to consider:
   (A) hardware
   (B) operating systems and other software
   (C) runtime errors
   (D) all of the above

5. Quality Assurance:
   (A) is an unnecessary intrusion for most programmers
   (B) aims to exceed customer expectations, which leads to higher profits
   (C) is a government requirement for all software development
   (D) is only relevant to large businesses

6. Which term refers to protecting yourself when communicating online?
   (A) privacy
   (B) cyber safety
   (C) firewall
   (D) anti-virus software

7. Usability testing aims to:
   (A) ensure software performs its functions correctly
   (B) maximise the efficiency and satisfaction of users
   (C) reduce the number of errors
   (D) make users feel they are part of the development process

8. Codes of conduct for software developers are usually produced by:
   (A) governments
   (B) professional associations
   (C) user groups
   (D) computer manufacturers

9. Which term best describes software that causes some undesirable or uninvited process?
   (A) bug
   (B) virus
   (C) malware
   (D) spyware

10. The responsibility for distributing viruses with software products lies with the
    (A) software publisher
    (B) software retailer
    (C) software development business
    (D) creator of the virus

11. Define the following terms:
    (a) malware (c) social networking (e) cyber safety (g) code of conduct
    (b) ergonomics (d) intellectual property (f) inclusivity (h) privacy

12. Make up a list of responsibilities software developers have to their end users. Are these responsibilities also rights for the end users? Explain.

13. Responding to software problems is an important responsibility for software businesses. Choose TWO software products with which you are familiar. For each product describe appropriate types of software support that should be provided.

14. You are in charge of the development of a new multimedia product to educate primary students about high school life. Name and describe the tasks for each contributing author associated with the development of this product. Which of these authors would be entitled to royalty payments if the product were to be sold commercially?

15. A product has been developed which looks and behaves remarkably like a competitors product. Using the example code of conduct from the text, list the items in the code of conduct that have been breached. Explain each item you have included.

SOFTWARE PIRACY AND COPYRIGHT

Software piracy is the illegal copying and use of software. This occurs when the intellectual property rights of the developer/author are infringed. Software piracy can occur when an individual illegally installs software on his or her own machine. On a grander scale, software piracy can occur when unscrupulous businesses distribute copies of copyrighted software for financial gain.

The legal system provides certain rights and protections for owners of property. Intellectual property is property resulting from the fruits of mental labour. The owner, usually the creator, can protect their intellectual property rights using the copyright laws. Often a licence agreement is used to formalise the contract between the intellectual property owner and the purchaser. Licence agreements help enforce copyright laws and in addition they routinely include details of warranties and limited liability clauses. A warranty protects the purchaser and limited liability clauses protect the seller.

Advances in digital technologies, particularly in regard to networks, have made ascertaining the ownership of intellectual property rights an often complicated task. The establishment of high-speed networks has made it possible for software (as well as other digital works) to be collaboratively developed and then delivered to scores of individuals with just a few keystrokes. Modules of software code can be combined and used by others in new projects. A whole new legal area based on ascertaining who owns the intellectual property rights to a piece of software is emerging.

The situation with software piracy and copyright becomes complex when we consider the differing methods of software distribution and licensing that exist within the industry. In this section, we will discuss a number of concepts associated with piracy and copyright. We conclude by looking at copyright laws and licence agreements.

PLAGIARISM

Plagiarism is appropriating or imitating another's ideas and manner of expressing them and claiming them as your own. When somebody passes off someone else's creation as his or her own, they are plagiarising that person's work.



Consider the following:

Imagine you are doing a school project on wombats and you merely copy a project you find on the Internet. You hand in the project to your teacher claiming the project is your original work. Making the claim that the project is your own work is considered plagiarism.

In terms of software, the plagiarism situation can become less clear. Many people and products are involved in the creation of new software. Often code is modified to suit a new task. The main point is that any contribution to a project needs to be acknowledged and work obtained from other sources should never be said to be your own.



GROUP TASK Discussion

The vast amount of information available on the Internet is incredible. Discuss the ethics of copying information from the Internet. When does legitimate research on the Internet become plagiarism?

COPYRIGHT LAWS

Copyright laws are in place to safeguard the legal intellectual property rights of authors of any original works. The purpose of these copyright laws is to provide economic incentives for creative activity, hence promoting the progress of creative endeavour. Copyright protects the expression of ideas rather than the ideas themselves. As the word copyright implies, the copyright laws give the owner the sole right to reproduce the work.

The current copyright act in Australia was written in 1968. This act has been amended numerous times. New technologies, in particular software, have been included as amendments to *The Copyright Act 1968*. Most countries have copyright acts that now cover software. Generally, the laws covering copyright for a particular country are enforceable in other countries.

A significant amendment to Australian Copyright Law was passed in 2006 (*The Copyright Amendment Act 2006*). This amendment was largely to bring Australian law into agreement with United States copyright law as a consequence of the signing of the free trade agreement between the two nations. Major additions included making it illegal to remove technical copy protection measures and making it legal to make copies of material for person use. For example it is now legal to record television and radio for viewing at a later time and it is legal to copy music on a purchased CD onto a computer or portable device. It remains illegal to distribute these recordings or copies to others.

Following is a summary of the general laws of copyright followed today by most nations.

Copyright can be described as a collection of rights, which include the right:

- to reproduce the copyrighted work.
- to make derivative works (e.g. a movie from a novel).
- to distribute copyrighted works to the public.
- to perform certain works in public (e.g. music).
- to display certain works in public (e.g. paintings).

Common amendments to copyright laws particular to software:

- One copy may be made for backup purposes only.
- When ownership of a licence is transferred, all copies must be handed over or destroyed.



Backup
A copy of software or data made to guard against accidental loss of the original.

- Decompilation and reverse/backward engineering are not permitted. However, a product may be decompiled or reverse/backward engineered to provide information to other developers in regard to their product interfacing with the copyrighted product. (Note: a special amendment existed to allow for decompilation to overcome the perceived effects of the Year 2000 bug).

Copyright commences upon creation of a work and lasts until the author's death plus a further 70 years. Copyright exists regardless of whether the work has been registered with the copyright office. However, works must be registered before legal proceedings can commence for copyright infringements.

The familiar copyright symbol © is not actually required but is recommended. The law, as it currently stands, assumes copyright exists unless the author has explicitly stated that a work is not under copyright. Copyright of a work can be transferred to others. This must be done in writing to ensure validity.

The licence agreement included with most commercial software is designed to implement the copyright laws for that particular product. By agreeing to the licence terms, you are entering into a legal contract with the copyright holder.

CLASSIFICATION OF SOFTWARE IN TERMS OF COPYRIGHT

There are four, commonly used classifications:

- Commercial
- Open Source
- Shareware
- Public Domain

The restrictions and limitations regarding these classifications are different.

Commercial

Commercial software is most software purchased from software publishers, computer stores and other retailers and software distributors. You are actually purchasing a licence to use the software. The licensing company owns the product and owns the product's copyright.



Decompilation
The opposite of compilation. Translating machine executable code into a higher-level code (usually assembler). This allows the program's design to be more easily understood.

Generally, commercial software licences state that:

1. The software is covered by copyright.
2. One archival copy may be made for use only if the original fails.
3. Modifications to the software are not permitted.
4. Reverse engineering and decompilation are not allowed.
5. Use of the work as part of another package is not allowed without permission.



Reverse/Backwards Engineering
Analysing a product and its parts to understand how it works and to recreate its original design, usually with the purpose of creating a similar product based on this design.

Open Source

Although covered by copyright law, open source licences, such as the popular GNU GPL (general purpose licence), specifically remove many traditional copyrights. The source code is developed collaboratively and is available to all to modify and redistribute. The only significant restrictions being that the author is recognised and that modified products must be released using the same unrestricted open source licence. The aim of open source licences is to ensure users can freely use and modify software without fear of legal challenge. This encourages collaboration and encourages sharing of ideas within the software development community. For most users software distributed under an open source licence can be installed and used without restriction.

Open source licences and in particular the GNU GPL are based on the premise that nobody should be restricted by the software they use. The foundation of the GNU GPL is based on the following four freedoms that every user should have:

- the freedom to use the software for any purpose,
- the freedom to change the software to suit your needs,
- the freedom to share the software with your friends and neighbors, and
- the freedom to share the changes you make.

Open source licensing encourages the collaborative development of software. There are a large variety of popular and widespread software applications developed as open source products. Some examples include the Linux operating system in its many different forms and versions, the MySQL database management system, Apache web server and also a variety of different programming languages such as Java, PHP and Perl.

Shareware

Shareware is covered by copyright. As with commercial software, you are acquiring a licence to use the product; you do not own it. Purchasers are allowed to make and distribute copies of the software. Once you have tested the software and decided to use it, you must pay for it. Distributing software as shareware is essentially a marketing decision; purchasers become distributors of the product. The legal requirements in regard to copyright still exist.

In general, shareware licences state:

1. The software is covered by copyright.
2. Copies can be made for archival or distribution purposes.
3. Modifications to the software are not permitted.
4. Reverse engineering and decompilation are not allowed.
5. Use of the work as part of another package is not allowed without permission.

Public Domain

Software becomes public domain when the copyrights holder explicitly relinquishes all rights to the software. Software in the public domain must be clearly marked as such. Just because a product does not bear a copyright symbol does not mean that it is not covered by copyright.

For public domain software:

1. Copyright has been relinquished.
2. Copies can be made with no restrictions.
3. Modifications to the software are allowable.
4. Decompilation is allowed.
5. Developing new works based on the product is allowed without restriction.



GROUP TASK Discussion

Open source and public domain software is free. Given the widespread acceptance of many of these free software products one might presume that commercial and shareware will no longer be economically viable. This is clearly not the case. For example most users (and organisations) continue to use commercial operating systems and applications, such Microsoft Windows and Microsoft Office. How can you explain the continuing use of commercial and shareware software? Discuss.

OWNERSHIP VERSUS LICENSING

Software obtained from outside sources is not generally owned by the user. The software developer who is the author of the product retains ownership of the product. This is the same as printed books such as this textbook. As the purchaser of this book, you are licensed to use this book but you cannot copy it or distribute copies. Some software and printed media are in the public domain and may be freely copied and distributed (see above section on public domain). It is important to read the licensing agreement before installing new software to be sure you are using the product legally.

You own any software and data files you create. As a software developer, it is up to you to create a legal licence agreement to accompany any product you distribute. This licence agreement will be the basis of the contract between you as the software developer and the end purchaser.



GROUP TASK Discussion
There are other products and services apart from software where licensing is used rather than ownership. Create a list of some examples.

SOFTWARE LICENCE AGREEMENTS

Software licences are intended to enforce the intellectual property rights of software developers. These licence agreements are enforceable by law, including copyright laws. Thus, the terminology used must be legally correct if the agreement is to stand up to the scrutiny of the courts. As well as protecting the intellectual property rights of software developers, licence agreements also protect developers from legal action should their products result in hardship or financial loss to purchasers.

Licence terminology

Because of the legal nature of licence agreements, they often use terminology that is difficult to understand. As licence agreements result in a legal contract between the purchaser and the seller, it is important to have an understanding of the terminology.

- Licence – Formal permission or authority to use a product. In relation to software, licences are almost always non-exclusive; this means the product can be licensed to multiple users. The license does not give users ownership of the software, rather they are granted the right to use the software.
- Agreement – A mutual arrangement or contract between parties. Acceptance of a software licence agreement can be made in various ways. Products downloaded from the Internet often require clicking ‘OK’ to on-screen terms and conditions. Installing pre-packaged software indicates acceptance of the agreement. Specialised and custom solutions usually require the signatures of both parties to legalise the agreement.
- Term – The period of time the agreement is in force. In most cases, the licence agreement commences immediately the terms and conditions have been accepted. The agreement remains in force for as long as the terms and conditions are upheld. Normally either party can terminate the agreement if the other party fails to act according to the terms and conditions within the agreement.
- Warranty – An assurance of some sort - a guarantee. Software products normally contain limited warranties. They may guarantee the medium (i.e. CD-ROMs) work correctly. However, most warranties will state that the product is sold ‘as is’. This means any bugs in the product, or if the product does not meet the user’s needs, is not covered by the warranty. For custom-built large-scale applications

this is often not acceptable and may be replaced by statements limiting the software developer's liability should a problem occur.

- Limited use – Software licences do not give purchasers unrestricted use of the product. Commonly usage of software products is restricted to a single machine. Copying of the product is not permitted unless for archival or backup purposes. The product cannot be altered or modified.
- Liability – An obligation or debt as a consequence of some event. Licence agreements normally restrict the liability of the software developer to replacing the product or refunding the purchase price should an error or other problem occur.
- Program – Refers to the computer software. This usually includes both executable files and included data files. The program does not include the media; it refers to the software stored on the media.
- Reverse engineer – In terms of software, this usually means the process of decompiling the product. Most agreements do not allow licensees to reverse engineer their products. This protects the intellectual property rights of the software developer.
- Backup copy – A copy of the software made for archival purposes. Backup copies should only be used in the event of the original media failing. If software is resold or the licence agreement terminated then backup copies should be destroyed.



Consider the following:

Most software products are now downloaded from the internet. Prior to installation the user is required to accept the terms of the software licence agreement by clicking on an "Accept" button. This is known as a clickwrap (or click-through) licence.



GROUP TASK Discussion
Examine a clickwrap licence and identify areas within the licence which deal with each of the dot points above. Are all the dot points above relevant to clickwrap licence agreements?



Consider the sample *Software Licence Agreement* that follows:

IHT is a software development company specialising in automating hotels or group of hotels. The appendices of this agreement are not reproduced (they explain the actual software and support applicable to a particular hotel or group of hotels).



GROUP TASK Identify
This agreement allows the software to be installed at the named hotel, with a number of restrictions. Identify the restrictions mentioned in the agreement in regard to installing the product.



GROUP TASK Discussion
Does the agreement allow hotels to alter the software?
If the software is altered, what are the conditions in the agreement in regard to correcting these modifications?



GROUP TASK Discussion

The software will be regularly updated and modified. Explain the nature of maintenance and support provided to customers in this agreement. What are the responsibilities of the customer in regard to updates?

SOFTWARE LICENCE AGREEMENT
Between
INTEGRATED HOTEL TECHNOLOGIES PTY. LTD. (IHT) and

Hotel Name: _____

Address: _____

Number of Rooms: _____

STANDARD TERMS AND CONDITIONS

1. Definitions. As used in this Agreement, these Standard Terms and Conditions, and all Appendices, Exhibits and Amendments attached hereto, the following terms are defined as follows:
   - A. Agreement. "Agreement" refers to the Agreement for IHT Software Licence, Training and Sale or Data Processing Machines, all Appendices and Exhibits attached hereto which are executed by IHT and Customer.
   - B. Appendices. "Appendices" means the Appendices, which fully define the Application Software, services, materials and maintenance, provided as follows:

Appendix A	Front Office System	Appendix C	Maintenance
Appendix B	Interfaces	Appendix D	Additional Terms and Conditions

   - C. Application Software. "Application Software" means a computer program that is designed to automate a particular function within a hotel (e.g., front office.).
   - D. Customer. "Customer" means an IHT Customer who is an end-user of the System. They do not have the right to grant further sublicences with respect to the System.
   - E. CPU. "CPU" shall mean the Central Processing Unit or units located at the place of business designated as "Hotel Name" on this Agreement, which is the unit of the computer hardware that controls the interpretation and execution of program instructions.
   - F. Hotel. The term "Hotel" shall mean a single building or group of buildings in close proximity to one another operated as a single entity under a single name.
   - G. Object Code. "Object Code" means the version of the Application Software in which the Source Code language is converted or translated into the machine language of the computer with which it is to be used.
   - H. Machine. "Machine" means the tangible data processing computer equipment.
   - I. System. The "System" means all object code for the Application Software, including documentation, enumerated in the Appendices hereto.
   - J. Source Code. "Source Code" means a computer program written in any of several programming languages, understandable to humans, in which a computer is given instructions.
   - K. Use. "Use" shall mean copying or transmitting any portion of the Application Software described in the attached Appendices into the Central Processing Unit located at the hotel address indicated on this Agreement, for processing instructions or statements contained therein.
2. Title: Functional Representations. IHT represents that it has sufficient right, title and interest in the System to grant this licence. IHT represents that the System will function as described in the attached Appendices. IHT representations under this paragraph extend to the CPU type designated herein provided such CPU has not been modified or otherwise altered by Customer, Hardware Manufacturer, or any other Party from Hardware Manufacturer's standard configuration for such CPU and operating system. Title to all Systems shall at all times remain in the name of IHT.
3. Use of the System. The Customer agrees that the right to use the Application Software indicated in the Appendices extends only to the specified CPU(s) at the location and for the hotel indicated above. If Customer desires to use any portion of the System on an additional CPU or CPUs, additional agreements must be negotiated. The Customer's right to use the System is expressly made subject to the Customer's performance of its obligations under the terms and conditions of this Agreement. The Customer's failure to perform any such obligations shall constitute a breach of this Agreement and give rise to IHT's right to terminate this Agreement. Customer may copy the Application Software in whole or in part for use on the designated CPU at the designated location for archival purposes only, provided that no more than one copy of the Application Software is in existence at any one time without the prior written consent of IHT. The Application Software, and any original or copies thereof, in whole or in part, are and remain the property of IHT, irrespective of the ownership of the media on which the software is contained. System documentation may not be copied or reproduced; copies are available from IHT at the standard rate then in effect.
4. Obligations of Customer. Customer shall be solely responsible for and shall bear all costs associated with the goods and services listed herein. Failure to make timely payments for all goods and services, including payment of all fees for licences and services, as specified herein, constitutes a breach of this Agreement.

5. Delivery of System. Unless personally delivered at IHT corporate office, IHT shall arrange for shipment of the System to the hotel specified on page one. All other forms of shipment shall be at Customer's request and sole expense.
6. Term. This Agreement is effective from the date this Agreement is accepted by IHT. The duration of this Agreement is two years, which shall be renewed thereafter on an annual basis at no additional cost to Customer, until terminated. The term for the maintenance services is one year, renewable on a mutually agreed upon basis.
7. Transfer of System Licence. The Customer may not transfer this licence without the prior written consent of IHT. The Third Party must purchase or otherwise acquire the right to operate the System at the location for the hotel indicated above. All other transfers are void and deemed to be in breach of this Agreement.
8. Installation and Training. This Agreement includes Installation and/or Training if so indicated on Page 1. Any additional training requested by the Customer will be billed to the Customer at IHT standard rate then in effect for such services.
9. Travel Expenses. The Customer shall reimburse IHT for all reasonable expenses incurred by IHT employees, Agents or Representatives while travelling between IHT offices and the Customer's hotel, as well as reasonable expenses incurred while providing installation, training, or support services at Customer's hotel. IHT will promptly bill the Customer, and the Customer agrees to pay IHT for all expenses not directly paid by the Customer.
10. Program Maintenance and Support. During the term of this Agreement, IHT's sole obligation regarding maintenance and support shall be limited to repair of any program or Operator Control Language (OCL) procedure defects that prevent normal use of the System. This shall be at no additional cost to the Customer. The Customer shall give IHT prompt notice of any alleged program or OCL procedure defect. Customer agrees to fully cooperate with the diagnosis and repair of any such program or OCL procedure defect. IHT's obligations shall be waived in the event of:
    A. Program or OCL procedure defects which are the result of improper handling or use; or
    B. Program or OCL procedure defects which have been previously reported to the Customer and for which program or OCL procedure corrections have been submitted to the Customer; or
    C. Any problems with data on media which have been caused by deficiencies in Hardware Manufacturer Program Products, or by improper handling or use by the Customer; or
    D. Program or OCL procedure defects as a result of the use of CPU, peripheral equipment including, but not limited to, terminals, tape drives, printers, emulators, scanners, and systems software.
    E. An unauthorized alteration or revision to the System. The Customer shall reimburse IHT for all reasonable expenses incurred, travel time and actual time expended by IHT employees, agents or representatives in responding to fake maintenance or support claims or in repairing any unauthorized alterations or revisions to the System, including items (A) - (D) above, in accordance with IHT's published rates then in effect, set forth separately, and incorporated herein by reference. IHT further reserves the right to refuse to make repairs resulting from or required by any unauthorized alterations or revisions to the System.

    If IHT corrects defects or problems to the System attributable to OCL error or programming modifications made by Customer, Customer agrees to pay IHT the standard rates for time and materials in accordance with the IHT published Rate Schedule.
11. Hardware, Hardware Manufacturer Software and Hardware Maintenance. Acquisition of the necessary Hardware, Hardware Manufacturer Software and Hardware Maintenance Agreements necessary to use the System shall be the sole responsibility of the Customer. Unless such materials and services are acquired through IHT, IHT will not assume any responsibility for the Hardware, software or maintenance, or their use by the customer.
12. Modifications, Enhancements and Updates.
    A. The Customer may not modify, enhance or change in any way the System supplied by IHT pursuant to this Agreement, without prior written consent from IHT. Any unauthorized change shall be deemed to be a material breach of this Agreement. Any authorized or unauthorized modification, enhancement or other change shall immediately be delivered to IHT and shall become the sole property of IHT.
    B. IHT may from time to time modify, enhance or otherwise change the System without notice to the Customer. Provided the Customer is current on the Maintenance fee for the respective Application software, IHT will immediately ship updated software to the Customer at no cost to the Customer. Such updates and enhancements may include, by way of example and not by way of limitation, revised report formats, new and generally desirable reports, changes to screen layouts, and such items as IHT determines may contribute to the general marketability of the System. Customer must install the updated System within 60 days of receiving the update software. Failure to install such updates will void IHT's obligations under the Maintenance provisions contained herein.
    C. Modifications or enhancements which IHT separately prices and offers to its other Customers for a specified price may be offered to the Customer at the price specified in IHT's standard rate schedule then in effect.

Fig 1.16
Sample software licence agreement.



HSC style question:

MineSoft is a software development business that develops, distributes and installs various commercial software applications to the mining industry. The software licence agreement accompanying each of MineSoft's products includes the following clause.

> *13. LIMITATION OF LIABILITY. Should you encounter problems with this software product and MineSoft is found to be at fault then MineSoft agrees to reimburse you the actual value of damages up to the amount you paid MineSoft for the software product.*

Discuss the appropriateness of this clause in terms of the rights and responsibilities of software developers.

Suggested Solution

There are a number of relevant issues with respect to the developer refunding the client up to the purchase price of the product.

If the software application modifies the operating system instructions or contains a virus, or in some way leaves the purchaser vulnerable as a result of the installation of this software, it would be reasonable to expect that the developer accept the responsibility for this. In the same way, if the application calculates values that are critical such as the amount of tax to be paid, or dosages of medicines, for example, and these are calculated incorrectly by the application, clearly the developer should accept the responsibility for this, and the amount to be paid to the client should not be limited just to the refund of the purchase price.

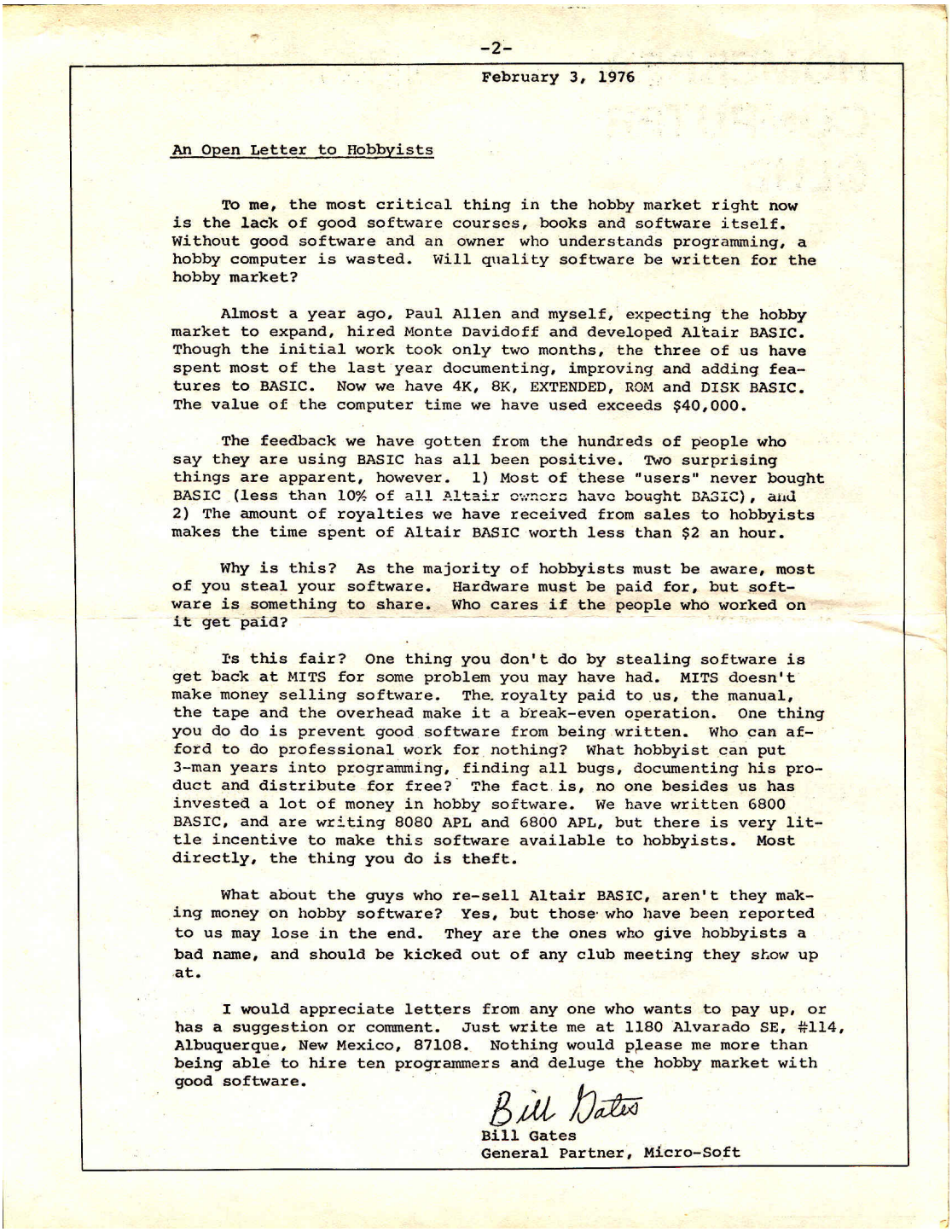
On the other hand, the developer has no control over the environment under which the client will be running their software (including the operating system or hardware used) and therefore cannot be held responsible for issues that become apparent in particular environments. It is assumed that the developer has fully tested their product before distributing it, and that therefore any subsequent issue would not be their responsibility. Having said that, it is often in the interests of goodwill for the developer to look into such issues and advise and fix it for them if at all possible.

It is often the case that licence agreements are the final "fallback" position should legal action be taken. In many cases the developer will spend time and money to help their customers and correct the product. This makes sense for both ethical and marketing reasons.

CURRENT AND EMERGING TECHNOLOGIES USED TO COMBAT SOFTWARE PIRACY

Without some form of protection, software is simple to copy and it is virtually impossible to determine that a copy has been made. This is not the case with physical products, where the effort required to duplicate the product far outstrips the purchase cost. This is why software piracy is such an issue and also is the reason software developers implement various technologies to prevent piracy. Software piracy has been around since the introduction of computers. It was 1976 when Bill gates first encountered piracy. His early versions of BASIC were being stolen and distributed to

other early computer hobbyists. Gates' "An Open Letter to Hobbyists" written in 1976, is reproduced below in *Fig 1.17*.



-2-

February 3, 1976

An Open Letter to Hobbyists

To me, the most critical thing in the hobby market right now is the lack of good software courses, books and software itself. Without good software and an owner who understands programming, a hobby computer is wasted. Will quality software be written for the hobby market?

Almost a year ago, Paul Allen and myself, expecting the hobby market to expand, hired Monte Davidoff and developed Altair BASIC. Though the initial work took only two months, the three of us have spent most of the last year documenting, improving and adding features to BASIC. Now we have 4K, 8K, EXTENDED, ROM and DISK BASIC. The value of the computer time we have used exceeds $40,000.

The feedback we have gotten from the hundreds of people who say they are using BASIC has all been positive. Two surprising things are apparent, however. 1) Most of these "users" never bought BASIC (less than 10% of all Altair owners have bought BASIC), and 2) The amount of royalties we have received from sales to hobbyists makes the time spent of Altair BASIC worth less than $2 an hour.

Why is this? As the majority of hobbyists must be aware, most of you steal your software. Hardware must be paid for, but software is something to share. Who cares if the people who worked on it get paid?

Is this fair? One thing you don't do by stealing software is get back at MITS for some problem you may have had. MITS doesn't make money selling software. The royalty paid to us, the manual, the tape and the overhead make it a break-even operation. One thing you do do is prevent good software from being written. Who can afford to do professional work for nothing? What hobbyist can put 3-man years into programming, finding all bugs, documenting his product and distribute for free? The fact is, no one besides us has invested a lot of money in hobby software. We have written 6800 BASIC, and are writing 8080 APL and 6800 APL, but there is very little incentive to make this software available to hobbyists. Most directly, the thing you do is theft.

What about the guys who re-sell Altair BASIC, aren't they making money on hobby software? Yes, but those who have been reported to us may lose in the end. They are the ones who give hobbyists a bad name, and should be kicked out of any club meeting they show up at.

I would appreciate letters from any one who wants to pay up, or has a suggestion or comment. Just write me at 1180 Alvarado SE, #114, Albuquerque, New Mexico, 87108. Nothing would please me more than being able to hire ten programmers and deluge the hobby market with good software.

Bill Gates

Bill Gates
General Partner, Micro-Soft

Fig 1.17
Bill Gates' "An Open Letter to Hobbyists"



GROUP TASK Discussion

In the above letter Bill Gates is trying to appeal to hobbyists to be ethical. At the time, technologies to combat piracy didn't exist and laws were such that it was unclear if software piracy was even a crime. Given the current size, dominance and aggressive business tactics of Microsoft do you think a similar appeal, if made today by Bill Gates, would be justified? Discuss.

Over the years a variety of different technologies and strategies have been used in an attempt to minimise the possibility of piracy. Prior to the internet various anti-piracy strategies were attempted, such as the use of non-copyable data sheets and various forms of disk protection. The widespread use of the internet has made piracy a much more significant issue as software is distributed with ease across the globe. However the internet has also provided the means for software publishers to digitally verify legitimate users of their products. Technologies such as registration codes, serial numbers, encryption keys and back-to-base authentication all aim to verify the user possesses a legal copy of the software product.

Non-copyable datasheet

Commencing in the 1980s non-copyable datasheets were included with some software products. The user needed to enter codes (or answers) from the datasheet to continue using the software. For example, Broderbund software released the game "Where in the World is Carmen Sandiego?" in 1985. The game included a copy of the book "The World Almanac and Book of Facts" published by Pharos Books. During game play the user was required to answer specific questions whose answers could only be found within the accompanying almanac. As the almanac was a large volume it was a significant effort for potential pirates to copy the book – particularly when scanning technology was not widely available. Some other software products included datasheets printed using inks that could not be copied using the photocopiers commonly in use at the time.

Disk Copy Protection

Prior to the widespread use of the internet most software was distributed on floppy disks and then on CD-ROM. A variety of different technologies were utilised to prevent these disks from being copied. For example, some copy protected disks included data on areas of the disk that were not normally able to be read or placed the data on specific sectors of the disk. Most copying software at the time was unable to replicate these features and hence copying the disk was not possible. Over time hackers identified the disk protection methods being used and created software that could either disable the protection system or replicate the disks precisely. New disk protection technologies continue to be developed, however historically all have been cracked fairly rapidly.



GROUP TASK Research

Games consoles, such as PS3, Wii, Xbox 360 and Nintendo DS, use their own proprietary disk or cartridge formats. Although this provides significant protection against piracy it does not solve the problem. Research to determine piracy issues related to games consoles.

Hardware Serial Numbers

A variety of hardware components within every computer include an embedded serial number which cannot be altered. Software can be designed so that it examines these serial numbers and if they don't match then the program will not execute. In effect the software is linked to a particular machine. Many industry specific software products utilise this type of copy protection. Often these software products are sold together with the hardware as a complete system. On a Windows machine, the serial number of the computer can be determined by entering the following command at a command prompt `wmic bios get serialnumber`. This serial number can be retrieved as the software commences execution and compared to the stored serial number within the executable code. If they do not match then the software will not run.



GROUP TASK Discussion
During initial activation of the Window's operating system details about the hardware on your machine are linked to your copy of Windows. This information allows Microsoft to validate whether Windows is genuine. Non-genuine installations are unable to receive updates from Microsoft. Discuss advantages and disadvantages of Microsoft's validation system.

Site Licence Installation Counter on a Network

In larger organisations software is often installed from a network server. The organisation purchases a site licence which specifies the maximum number of machines who may either install or simultaneously execute the product. The network server includes a utility which monitors the number of times the software has been installed or the number of times the software is currently executing. For installation based site licences the server refuses to provide the installation files if the licensed number of installations has already occurred. If the site licence specifies the number of simultaneous users then each machine must gain permission from the network server before the software will execute. The server maintains a count of the number of current users. This counter is incremented as new users execute the product and it is decremented as users exit the product. Some multi-user applications execute on the server with users accessing the application via a client (often a web browser). In this case the application is executing once on the server and is able to track the number of clients to ensure the terms of the site licence are not exceeded.



GROUP TASK Research
Research to determine the functions of a site licence management application, such as Adobe License Manager. Make a list of advantages for the installing organisation and advantages for the software publisher.

Registration Code

A registration code is used to activate software products during the initial stage in the installation process. For single software licenses the registration code is unique to your installation. Site licenses commonly share a single registration code and also provide some facility for automating the activation process for multiple machines. This allows the installation to be performed without user intervention.

There is a wide variety of techniques used to verify registration codes. Some software applies an algorithm to the registration code so that it can be verified as correct. Although this prevents people entering random codes it does not prevent registration codes being used by others. Today most registration codes are either generated over the internet or are verified over the internet.

If the registration code is generated over the internet then the software developer will have access to some identifying information from the purchaser. For instance, an IP address, the person's email address, or even a serial number from the purchaser's computer. This information can be used to generate a specific registration code which will only activate the software if the other information is also present.

Other registration codes supplied with software must be verified over the internet for the software to continue operating. In many cases failure to verify results in reduced functionality or the software ceasing to function after a set period of time. In this case the verification process contacts a server which records the registration code. Therefore the system is able to detect if a registration code has already been used or in the case of site licences has been used too many times.



GROUP TASK Discussion
Although many computers are permanently connected to the internet there are many which are not. Therefore it is unreasonable for software to not install or cease to function simply because software cannot be activated or registration codes cannot be verified. Discuss.

Encryption Key

Encryption effectively scrambles the data or executable code in such a way that it is virtually impossible to make sense. The decryption code is required to reverse the encryption process. There are two common systems of encryption; single key and two key systems. Single key or secret key encryption uses the same key to both encrypt and decrypt the data. Two key systems use one key (called the public key) to perform the encryption and a different key (the private key) to decrypt. Public/private two key systems operate within the secure sockets layer (SSL) protocol used to perform most financial transactions over the internet.

Single key systems are difficult to implement over networks because the common key must be securely transmitted to the other party if the system is too be secure. Two key systems allow the encryption (or public key) to be shared with anyone as only the owner of the private key will be able to decrypt the message. For example (refer *Fig 1.18*), as the purchaser of a software application, your computer generates a

BEGIN Two Key Software Transfer
  Publisher computer requests public key from purchaser computer
  Purchaser computer generates public/private key pair
  Purchaser computer sends public key to publisher computer
  Publisher computer uses public key to encrypt software package
  Publisher computer transmits encrypted package to purchaser computer
  Purchaser computer decrypts package using private key
  Purchaser computer installs software package
END Two Key Software Transfer

Fig 1.18
Two key software transfer algorithm.

public/private key pair. The public key is transmitted to the publisher's computer which uses it to encrypt the software package. The package is sent to your computer which then uses your private key to perform the decryption prior to installation. The generation of the keys is performed automatically and transparently without the need for user interaction. Once the transfer is completed the keys are destroyed. It is also possible for a two key system to be used to securely transfer a single key encryption code. The actual package is encrypted using the single key and then the same single key (which was transmitted securely) is also used to decrypt the package.



GROUP TASK Research
Most two key (public key) encryption systems are based on prime numbers and the RSA algorithm. Research the general nature of RSA algorithms including their use of prime numbers.

Back-to-base Authentication

Back-to-base authentication means the application contacts the software publisher's server to verify the user or computer holds a valid software licence. If the licence is verified as correct then the software will execute. Back-to-base authentication can be used when installing the software or it can be used each time the software is run. For software which requires internet connectivity to perform its functions, back-to-base

authentication is an excellent strategy for preventing piracy. However for software whose functions do not use the internet back-to-base authentication restricts user access when an internet connection is not present.



Consider the following ways in which software can be pirated:

- End User Copying. Individuals make copies of software or lend software to friends and workmates.
- Piracy of OEM Products. Manufacturers of Personal Computers or Original Equipment Manufacturers, install operating systems and other applications illegally on new machines they sell. A licence for all software should accompany the newly purchased machine.
- Counterfeiting. Unscrupulous organisations copy original software and then on sell it as legal copies. Often the packaging and manuals are also included and on the surface, the product looks, if not exactly the same, then very similar to the bona fide article.
- Mischannelling. Software sold with discounted licences is resold as a fully licensed product. An example of this would be an academic version being retailed as a fully licensed version.



GROUP TASK Discussion
Do you think the methods outlined on the previous pages for combating software piracy are effective against end user copying, piracy of OEM products, counterfeiting and mischannelling? Discuss.



HSC style question:

(a) Outline the purpose of copyright laws.

(b) Describe the relationship between copyright law and software licence agreements.

Suggested Solutions

(a) Copyright laws are in place to acknowledge the time, effort and intellectual creativity put in to the development of software by software developers. By protecting their authorship through these laws, it encourages others to value the work of the developers, and to encourage others to develop new ideas and products without fear of being taken advantage of. Copyright law aims to afford similar rights to the ownership of intellectual property as are provided for ownership of physical property.

(b) A software licence agreement is a legal contract between the author of the software and the user which defines what a purchaser of the software licence may do with the software whose licence has just been purchased. These defined conditions (such as how many copies may be run concurrently, whether or not the software can be shared with others, etc.) are specified so there is no misunderstanding as to how the software may be used, in a way which protects the author of the software by ensuring that the intellectual property is appropriately acknowledged and protected by copyright laws.

SET 1B

1. Within a software licence agreement there is a clause which restricts the amount of money a user can claim if the software causes some issue. Which of the following best describes this clause?
   (A) warranty
   (B) limited use
   (C) limited liability
   (D) term

2. Falsely claiming ownership of someone else's work is:
   (A) piracy.
   (B) plagiarism.
   (C) counterfeiting.
   (D) theft.

3. The purpose of copyright laws is to:
   (A) provide incentive for creative endeavour.
   (B) protect the expression of ideas.
   (C) provide the owner with the sole rights to copy and distribute a work.
   (D) All of the above.

4. Software where users are encouraged to distribute copies but must pay the owner if they continue to use the product is:
   (A) commercial
   (B) shareware
   (C) open source
   (D) public domain

5. In terms of software copyright law, which of the following is FALSE:
   (A) One copy may be made for backup purposes.
   (B) Derivative works based on the original cannot be made.
   (C) Distributing copies is not permitted.
   (D) Backup copies may be retained if the licence is sold.

6. Reverse engineering is the process of:
   (A) studying the original design of the product.
   (B) analysing a product to determine its original design.
   (C) using a product as a major part of a new product.
   (D) translating executable code back into a higher-level language.

7. Software Piracy can be best described as:
   (A) theft.
   (B) copying software.
   (C) taking from the rich.
   (D) infringing copyright.

8. The only classification of software where copyright has been relinquished is:
   (A) commercial.
   (B) shareware.
   (C) open source.
   (D) public domain.

9. Which of the following is TRUE:
   (A) Works must be registered with the copyright office before copyright laws will cover a work.
   (B) The copyright symbol © must be displayed on all copyrighted works.
   (C) Transfers of copyright must be in writing to be legally valid.
   (D) Copyright laws are designed to safeguard publishers.

10. Selling an academic version of a product as a fully licensed version is known as:
    (A) mischannelling.
    (B) counterfeiting.
    (C) end-user copying.
    (D) piracy of an OEM product.

11. Define each of the following terms:
    (a) plagiarism
    (b) shareware
    (c) open source
    (d) public domain
    (e) reverse engineering
    (f) decompilation
    (g) backup
    (h) single key encryption
    (i) public/private key encryption
    (j) authentication

12. Explain the connection between intellectual property, copyright laws and licence agreements. Who is responsible for creating each of these?

13. Apparently, the majority of home users have illegal software installed on their computers. Discuss the ethics of this situation in terms of firstly, the home user, and secondly, the software developer.

14. Choose a large software development company with which you are familiar. For this company, research the methods they are utilising to combat software piracy.

15. A software product is to be distributed to customers via the Internet. Devise some strategies the distributor could implement to ensure the owner's copyright rights are protected.

USE OF NETWORKS

The widespread use of networks, including the internet, has revolutionised the development of software. The ability to collaborate and share resources greatly improves productivity. Although network speeds continue to improve they still provide a bottleneck for network based software. Changes to the user interface in terms of how data is entered and retrieved across the network can greatly improve response times. In addition security and privacy concerns (discussed earlier in the chapter) must be addressed when data is transferred across public and often insecure networks.



Network
A collection of computers connected electronically to facilitate the transfer of data and software.



Response Time
The elapsed time between a request (or user input) and a response (or output to the user) from the system.

Use of networks by software developers

Currently all but the smallest software applications are developed collaboratively by a team of developers. In many cases (particularly for open source development) the programmers have never met in person and may live in different countries. Without networks, in particular the internet, many of the applications we use on a daily basis would not exist. In addition to communicating with other developers many programming language libraries and related documentation reside on the web. During coding programmers are able to research and then in an instant download and incorporate existing source code into their application. Clearly both online collaboration and ready access to code and other resources greatly improves the productivity of software developers.



Consider collaborative development

The idea of collaborative development sounds terrific, however having multiple programmers all working on the same or different modules of source code can easily become a management nightmare. A variety of different versions of the same module can and do exist and determining which is superior is not clear. Furthermore each developer requires access to current versions of related modules, yet these related modules may well be currently undergoing their own revisions.



GROUP TASK Discussion
Brainstorm a list of issues that are likely to be encountered when multiple developers are working on a single software project. Can you suggest strategies to help manage these issues?



GROUP TASK Research
There are a number of software tools available to automate the management processes needed to collaboratively develop software. Git is one such product, and is itself an open source collaboratively developed project. Research a collaborative development software tool and list its significant functions.

Use of networks by users when using network based software

No doubt we've all accessed a web site, waited and then given up because the response time is too slow. Even sites that generally respond quickly can experience poor response times during periods of high activity. For software developers response times can be difficult to control with certainty. The network conditions and server load are often beyond the software developer's control. For instance, the slowest network link in the transmission determines the speed of the transmission, but for internet connections this is completely beyond our control. Nevertheless there are many design techniques which software developer's can use to make the most of the available network and server conditions.

Many RAD tools, including Flash, can create unnecessarily large files. Although media rich content can look amazing, it will have little impact if nobody can be bothered waiting for it to download. When embedded video and large images are included then response times will be severely compromised. For example, many websites use a single flash file with embedded video and images for an entire website, when not carefully designed such sites can take an eternity to load. Text based content is many times smaller in terms of file size compared to video and images. Although techniques such as compressing and streaming video can assist, often a link to such media is a more effective strategy.

Most network based software uses a client-server model. The client application is installed on the user's computer or runs within a web browser. Data or requests entered by the user into the client application are transmitted over the network to the server. The server application runs on the server and responds to the client's request by retrieving data from a database, which the server application then formats and transmits back to the user. Each client request and server response must travel over the network. Therefore we need to minimise the number of request and responses that applications perform and also minimise the quantity of data transmitted. Careful design of the order and number of inputs made by users before the data is sent over the network can yield significant performance improvements. Similarly minimising the amount of data retrieved by the server will also improve performance. Many commonly used network applications can provide some clues in this regard.



Consider the following:

- Consider searching for images using Google. The search criterion is entered in its entirety before being sent to Google's servers. The server responds with a limited number of small low resolution images. The user then selects one image at a time to view in higher resolution.
- A car insurance company provides an online quotation system. To determine the type of car, the user is presented with a list of all makes (Ford, Holden, Toyota, etc.). The selected make is sent to the server and a list of models is returned (Fiesta, Focus, Falcon, etc.). The user selects the model, which is sent to the server and a list of model variants is returned (G, GL, GLX, etc.).



GROUP TASK Discussion
Response times and user interface design are closely linked. Use the above two scenarios as examples to assist explaining the relationship between response time and interface design.

THE SOFTWARE MARKET



Marketing
The effort to determine and meet the needs and wants of current and potential customers.

The software market does have its own unique requirements compared to other industries. However, having said that, general marketing concepts still apply. One common misconception is that marketing and selling are interchangeable terms. This is not the case: marketing is a much broader area that includes sales as well as a great deal more.

Maintaining a software company's standing and position in the market requires that the company consider social and ethical issues when developing marketing techniques and strategies. The effect of not considering social and ethical issues will result in undermining the confidence of consumers to trust the assertions of software companies.

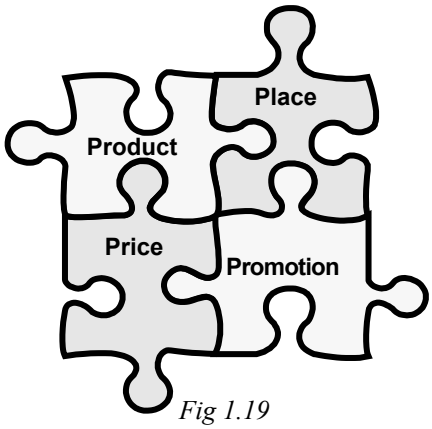


Fig 1.19
The Four P's of Marketing.

Marketing is often described in terms of the *Four P's of Marketing* namely, product, place, price and promotion. The relative proportion of resources allocated to each of these four efforts is often called the *marketing mix*. Let us consider each of these in terms of the software market with special consideration being given to the social and ethical responsibilities of those involved in the marketing process.

PRODUCT – WHAT DO YOU SELL?

Firstly, you are not so much selling a software product as selling an expectation of the product. The expectations of consumers must be realised in the actual product. It is unethical (and also poor marketing) for a product to not live up to the expected functionality desired by customers.



Consider the following scenario:

A desktop publishing application is advertised as having the ability to import graphics of different types. A potential customer purchases the product for their graphic design business with an expectation that they will be able to import graphics from their favourite photo editing software product. The customer installs the product, undergoes extensive training and then commences using it within their business. They discover that in fact the product is unable to import graphics of the desired format.



GROUP TASK Debate
The user manual contains details of the graphic formats that can be imported, and the one the customer requires is not mentioned. Debate the ethical issues arising for both the customer and the software company's marketing department.

Many software development companies specialise in writing custom software. These developers must ensure that the objectives of the system are realised once the final product has been developed and implemented. Not doing this makes no sense from a marketing point of view as well as from an ethical point of view.



Custom Software
A product designed to solve a particular problem or set of problems.

Products are often produced with insufficient knowledge of the requirements of the marketplace. The sales people are then confronted with the unenviable task of selling a product that does not conform to the requirements, and hence the expectations and needs of the potential client base. This can lead to sales people making exaggerated and false assertions in regard to the product's capabilities. This, of course, is unethical and in the longer term will result in industry dissatisfaction with the product and reduced sales. It is vital that customer needs, requirements and expectations are fully understood before software products are designed.

PLACE – WHERE DO YOU SELL IT?

The physical place where a product is sold is related to the type of product and its intended audience. Software is distributed in various locations using various methods. For example:

- Shop fronts, software retailers, e.g. computer stores, book stores.
- Industry specific distributors, e.g. businesses specialising in the sale of software to a particular industry.
- Direct sales, e.g. over the Internet, by mail order, direct mail.

The place where the software is sold can have an effect on the perceived ethical status of the product and the expectations of potential customers.

The software developer has a social and ethical responsibility to their distributors in regard to pricing. Often distributors will be given discounts for volume sales. These discounts can make it difficult for smaller distributors or distributors offering enhanced services to compete with the large volume distributors. It is the responsibility of the software company to ensure their pricing structure achieves a balance that is fair and equitable for all.



Consider the following:

A product that automates the storage and retrieval of legal documents for a law company is to be marketed. It would be inappropriate to sell this product through retail chains of computer stores. In fact attempting to sell via computer retailers could potentially reduce the perceived status of the product in the eyes of the legal profession. A company that specialises in legal software distribution would be a better marketer of this product. Although the product is the same regardless of the distribution method, the perceived ethical value of such software is enhanced when it is distributed using an industry specific distributor.



GROUP TASK Activity
There are numerous industry specific software applications. Make a list of any industry specific software used at you or your parents' place of work. Can you determine where this software is sold?

PRICE – WHAT DO YOU SELL IT FOR?

There are two commonly used methods for pricing a product. The first is based on a 'cost-plus' basis, and the second is referred to as 'consumer-based pricing'. The cost-plus method is relatively straightforward. Simply take the cost of producing the product and add the profit margin. Consumer-based pricing is not so straightforward. This method involves looking at what the consumer wants and how much they are prepared to pay for it. From both a marketing and ethical perspective, the second method makes more sense. By examining consumer needs and how much they are prepared to pay for that need to be satisfied, products will be produced which not only supply the consumer with an economical product but also enhance the social and ethical standing of the software development industry as a whole.

PROMOTION – HOW DO YOU SPREAD THE WORD?

Promotion consists of all the ways used to persuade people to buy products and services. Software marketers have an ethical responsibility about what they say and how they say it when selling to prospective customers. Advertising can be helpful to consumers who are investigating what is available in the marketplace. Promotional activities should help consumers to make informed choices. These advertising activities should provide accurate information that is inoffensive and not misleading.

Word of mouth is a powerful promotional technique. It is important to consider existing customers' needs as part of a promotional strategy. In regard to software, it is especially important to provide support both in terms of maintenance upgrades and *bug* fixes, as well as ongoing training support.



Consider the following:

A software company has developed a new computer game and is now in the process of developing a marketing strategy. Part of this strategy is a television advertising campaign using real life video footage with human actors depicting various scenes within the game. However, the actual game is entirely computer generated animation.



GROUP TASK Discussion
Discuss the ethical issues that arise with the television campaign being employed as part of the marketing strategy.
Is the consumer being given misleading information?
How great an influence would the television campaign have on the consumer's decision to purchase this product?

THE EFFECT OF DOMINANT DEVELOPERS OF SOFTWARE

In any market if one player's product dominants then it becomes the default purchase. This is largely what has occurred in the operating system market and also in the word processor and spreadsheet market. Microsoft's Windows family of operating systems is sold preinstalled on the large majority of personal computers and laptops. It requires effort on the part of the user to even purchase a computer without Windows. Furthermore most users are relatively familiar with Windows way of doing things, regardless of its performance or other limitations. Similar arguments are also true for Microsoft Office. The open source community has produced Open Office with similar functionality to MS Office but free of charge. Despite Open Office being free, MS Office continues to retain the majority of users.

Microsoft is not the only software development company that dominates - although its dominance in the operating system market is extreme. Apple has made a largely marketing decision when they include their operating system and other applications as part of any hardware purchase. This includes not only their computer products but also portable devices such as iPhones, iPads and iPods. In addition, for portable Apple devices software can only be purchased (or acquired) through Apple's iTunes App Store. Currently Apple receives a significant portion of all revenue (reportedly 30%) from every App sold.

There are also some surprising dominant players. Currently Apache dominates the web server market (with Microsoft's IIS a close second). Apache is an open source product and hence is distributed free of charge. The open nature of the internet is what made it possible for Apache to gain favour. Those who operate web servers are technically able and hence they are better positioned to ignore advertising and marketing campaigns in favour of functionality and performance. Apache's story demonstrates that in the software market (and largely due to the internet) it is possible for David to beat Goliath.



GROUP TASK Debate
"If a product is dominant within the market then it must have the best functionality and performance" Do you agree? Debate both sides.

IMPACT OF NEW DEVELOPERS AND SOFTWARE PRODUCTS

Although there are companies and products which dominate existing software markets there is still room for new players. Often new developers emerge due to inventiveness. That is, they invent a new software product that breaks new ground. Both Google and Facebook are prominent examples of companies which began with a bright idea. In early 1997 Google was nothing more than a Stanford University research project by Larry Page and Sergey Brin. By 2004 Google was handling some 85% of worldwide search requests. Facebook which officially began in 2004 was also a student project that took the world by storm. These are extreme examples and there are numerous others who have forged a successful living from new and innovative software.

Currently (2011), small software applications for mobile phones (known as Apps) are creating a significant market. Apps for Android phones and for Apple iPhones can sell in the millions. Otherwise unknown developers are able to access an enormous worldwide market using the Android Market (now owned and operated by Google) or through Apple's App Store. Most Apps sell for just a few dollars (and many are free) however when there are millions of potential customers a popular App can make significant profits. For example, the popular game "Angry Birds" currently has revenue exceeding $70 million for a very modest initial outlay. The "Angry Birds" development company Rovio Mobile is another business begun by university students – Niklas Hed, Jarno Väkeväinen, and Kim Dikert at Helsinki University Technology.

As China and other countries begin to lift their standard of living the market for software continues to grow at an enormous rate. The future for new and innovative software developers looks very healthy!



GROUP TASK Research
Research current examples of new and innovative software. List any inventive functions performed by the software and outline the history of the people who developed the software.

SET 1C

1. The effort to determine and meet the needs of current and potential customers is called:
   (A) Selling.
   (B) Marketing.
   (C) Promotion.
   (D) Researching.

2. Product, place, price and promotion are known as:
   (A) Marketing principles.
   (B) Marketing strategies.
   (C) The four P's of marketing.
   (D) The marketing mix.

3. A product is expected by consumers to have a particular feature. It in fact does not contain this feature. This is an example of:
   (A) a product failing to meet customer expectations.
   (B) false advertising.
   (C) unethical selling practices.
   (D) All of the above.

4. Which of the following is TRUE in terms of pricing:
   (A) Customers will always purchase a superior product, regardless of price.
   (B) 'Cost-based' pricing is more successful in terms of marketing.
   (C) Pricing should be based on perceived economic value for the consumer.
   (D) Discounting for volume sales will increase the overall volume sold.

5. A software distributor decides to sell its new product over the internet. Which of the following marketing elements are they considering?
   (A) Product.
   (B) Place.
   (C) Price.
   (D) Promotion.

6. Which of the following best describes response time?
   (A) The time taken for a user to understand the interface.
   (B) The time taken by a web server to respond to a request.
   (C) The time between a request and a response.
   (D) The response received by a timing device.

7. The proportion of resources allocated to each of the *Four P's of Marketing* is called:
   (A) Marketing principles.
   (B) Marketing strategies.
   (C) The four P's of marketing.
   (D) The marketing mix.

8. Which is NOT open source software?
   (A) Apache web server
   (B) MySQL database
   (C) Open Office
   (D) Angry Birds App

9. The best time to consider potential customers' requirements is:
   (A) when the marketing plan is being developed.
   (B) once the product has a large client base.
   (C) before the product is designed and produced.
   (D) during the testing of the product.

10. Software developers who develop custom applications:
    (A) must know their customers' expectations and requirements precisely.
    (B) need not be so concerned with the quality of their products.
    (C) do not require sales personnel to sell their products.
    (D) usually are more interested in the needs of their customers.

11. Name a dominant software application and its developer. Explain the effect of this dominant application on potential and actual competitors.

12. Imagine you are a small software developer. You have written an application that is of particular use to home users. Discuss how you would go about marketing this product yourself.

13. Choose a large software development company with which you are familiar. For this company, research the methods they are using to market their products. Can you suggest any improvements they could make to these marketing strategies?

14. Competition often has a significant effect on marketing. Do you think it is ethical to examine competitors' products before designing, producing and marketing new products? Explain.

15. A monopoly is when only one company sells a particular type of product - in other words, consumers have no choice. What do you think are the marketing consequences of a monopoly in the software development industry?

LEGAL IMPLICATIONS

As we have seen there are many significant social and ethical issues that need to be considered by those in the business of creating and distributing software. In this section, we consider a range of significant legal case studies both within Australia and also internationally.

Software implemented on systems throughout a country (or throughout many countries) can result in significant legal action if the software contravenes the law of the country in some way or its development does not comply with the legal contract between the software developer and the customer. The legal action may be the result of copyright breaches or can arise when software does not perform as intended.

NATIONAL LEGAL ACTION

Following are two well known cases that resulted in legal action within Australia:

- RACV versus Unisys
- NSW Tcard system

RACV versus Unisys

The RACV is a motor vehicle and general insurance company similar in nature to the NSW NRMA. In 1993 the RACV issued a Request for Proposal (RFP) also known as a Request for Tender (RFT) after making the decision to convert from its current paper-based claims management system to an electronic storage and retrieval system.

Unisys Systems won the proposal to supply RACV with their new system, one that would lead to more efficient handling of claims, reduced costs and an overall improvement of customer satisfaction.

During presentations of the new system to RACV, Unisys assured RACV that the response times and functionality shown during demonstration would be indicative of the live system. Acceptance testing was signed off in March 1995 and the project continued.

When the system was delivered however, the expected response times and functionality failed to eventuate. RACV ceased using their new system and reverted back to the paper-based system. Unisys' system would need to be redesigned. Even after a redesign, the new system failed to deliver on expectations and in fact was reported to be slower than the original paper-based system.

In June 1996, RACV terminated their contract with Unisys and sought damages through the courts citing false representation and misleading conduct with regards to the functionality of the system.

One critical issue that was to be a key-point in the court case was that the RACVs RFP was not included in the contract signed with Unisys. The RFP supplied by RACV stipulated that response times should be between a 2 and 4 second retrieval. Unisys argued that the contract that they signed, only stated a delivery of functional specifications, not response times as requested by the RFP.

The court over ruled Unisys arguments and awarded damages of $4 million to RACV stating that the RFP even though not part of the signed contract, were incorporated by necessary implication.



GROUP TASK Debate
Based on the above information (and your own research) make up a list supporting Unisys and another list supporting the RACVs case.

NSW Tcard system

The NSW Tcard system was a proposal put forward by the NSW State Government in 1996 to replace the automated ticketing system that was being used. It was hoped that the Tcard system, based on Smart Card technology would be operational in time for the Sydney 2000 Olympic Games.

The Sydney Olympic timeline passed however with various legal wrangling between parties preventing the start of development. Finally in 2002 the contract for the development of the system was finally awarded to the ERG Group and development commenced with a 2005 deadline given for full ticket integration.

The project was plagued by delays, with both the NSW State Government and the ERG Group blaming each other for failure to deliver on particular milestones. The deadline for a full commuter trial to take place was set for the end of 2004 but that deadline passed without the milestone being reached.

A trial by school students of the new technology was commenced in 2005 and by 2006 had expanded to include some private bus companies.

Still the delays continued and when the second deadline for commuter trials expired in 2007 without delivery being met, the State Government terminated the contract with the ERG group and sought damages of $95 million.

Following the termination of the contract, the school student Tcard system that had been in operation since 2005 was also brought to an end.

In its initial submissions, lawyers for the ERG Group allege that the NSW State Government refused to allow live testing of its system and that it had also stipulated that nothing could be installed until the system was 100% perfect. The ERG Group have indicated that this was one of the main reasons why the project failed.

The lawsuit has been set down to be heard by the end of this year (2011).

- ERG alleges that it did not receive sufficient cooperation from operators such as RailCorp and that they were not provided with reliable system testing reports throughout the development of the project.
- ERG has successfully developed smart card transport systems for Hong Kong, San Franciso, Stockholm, Singapore and a host of other countries.
- ERG have filed a counterclaim in the NSW Supreme court for $200 million for illegal termination of contract by the NSW State Government.



GROUP TASK Research
Research using the internet, to determine the outcome of the NSW State Government lawsuit. Also determine the status of ERG's counterclaim.



GROUP TASK Debate
Based on the above information (and your own research) make up a list supporting the NSW State Government and another list supporting ERGs case.



GROUP TASK Discussion
Is it reasonable for new software to be 100% perfect, particularly when no live testing has taken place? Discuss.

INTERNATIONAL LEGAL ACTION

Following are three well known cases that resulted in international legal action:

- Microsoft versus Apple Computer (GUI)
- Microsoft versus Netscape (Web browser)
- Search engines (Google versus national censorship laws)
- Napster versus Metallica (and RIAA)

Microsoft versus Apple Computer (GUI)

Apple computer released the Macintosh operating system with their range of Macintosh Personal Computers. This operating system was the first commonly available Graphical User Interface (GUI). Microsoft soon after released Windows, which in many ways looked and felt like the Apple Macintosh operating system. Both the Macintosh and Windows systems were distributed globally before any legal action began. Apple claimed that Microsoft had stolen their idea. After a long, drawn out court battle, a settlement was reached allowing both companies to continue to distribute their GUI operating systems.



GROUP TASK Discussion
Let's for a minute imagine that Apple Computer had won this case and Microsoft Corporation was ordered to withdraw the Windows operating system product from the global market.
Discuss the effects this decision would have had on Software Developers, Governments, Software Retailers and other businesses as well as the home computer user.



GROUP TASK Debate
With your knowledge of copyright law do you think that Apple Computer had a case to argue against Microsoft Corporation?
Debate both sides of the case and explain your reasoning for, firstly the Apple Computer side, and secondly the Microsoft Corporation side.

Microsoft versus Netscape (Web browser)

During 1990 to 1992, the Internet was primarily of academic interest. Most information was text-based being distributed via FTP and Gopher servers. It was not until 1993 that web browsers (and websites) as we know them today began to emerge.

The first widely available browser was Mosaic, the forerunner to Netscape. The first version was written for UNIX machines in 1993. Other browsers were under development such as the open source based Amaya, which is still in existence today. By the end of 1993, versions of Mosaic were available for Windows and Macintosh machines. At that time, there were some 500 web servers in existence, the first commercial site being Digital Equipment Company. In 1994, the web really took off and by the end of that year some 3 million servers were operating. Mosiac Netscape became a company with more than 80 percent of the browser market. Because of Netscape's educational roots, their browser was freely distributed to students, teachers and researchers.

In 1995, the potential of the World Wide Web was obvious and Microsoft could see this. Windows 95 was released with Microsoft Network (MSN) client software included, the aim being to establish the Microsoft Network as a parallel network to the

World Wide Web. MSN as a separate network was not successful and Netscape continued their domination. The release of Netscape Navigator 2 further secured Netscape's position. Navigator 2 included Email, frames, support for progressive JPEGs, Java support as well as support for SSL (Secure Sockets Layer) encryption. Microsoft's Internet Explorer 2 existed with a small yet significant following.

The battle for market share between Netscape and Microsoft began in 1996. Microsoft released Internet Explorer 3 and gave it away for free! Internet Explorer 3 had similar functionality to Netscape's product. Both Netscape and Microsoft were developing parallel technologies, each using their own standards. Web designers at that time often produced two versions of their site, one for Netscape browsers and one for Microsoft browsers. The World Wide Web Consortium (W3C) was supposedly the official standards body; however neither Netscape nor Microsoft adhered to these standards. In 1996, there were some 10 million web servers in operation.

The final crunch for Netscape came in 1997 when Microsoft packaged Internet Explorer 4 with Windows 95. Microsoft integrated the web into Windows 95 using the 'Active Desktop' concept. Netscape filed lawsuits against Microsoft, alleging Microsoft was taking illegal advantage due to their vast share of the operating system market. Microsoft's response was that Internet Explorer was an integral part of their operating system. No real resolution was ever reached.

In 1998, Netscape had lost most of its market share as a result of Microsoft's domination. Netscape's browser became an 'open source' product. Source code was released to the open source community as the Mozilla project. In 1999 America Online (AOL) purchased Netscape and then in 2003 AOL provided the initial funds to setup the Mozilla project. The Mozilla project now develops the popular Firefox browser together with a variety of other open source applications for the web.



GROUP TASK Debate
In your view were Microsoft's actions in regard to its marketing of Internet Explorer reasonable? Debate both sides.



GROUP TASK Discussion
How have web browsers altered the way in which we communicate? What makes web browsers such useful applications compared to other methods of communication? Discuss in relation to computer-based communication as well as other forms of communication.

Search engines (Google versus China's national censorship laws)

China has some of the most restrictive censorship laws in the world. Google on the other hand represents itself as being a bastion of free speech and free press. It was therefore somewhat inevitable that a Google search engine located in China would cause issues for both the Chinese Government and Google.

Google china first began operations in 2005 and in early 2006 google.cn was launched. The results from google.cn searches were filtered and censored to comply with China's local censorship laws and regulations.

In early 2009 China blocked access to Google's YouTube site and in January 2010 a number of US tech companies operating in China claimed to have been hacked. Google responded with an announcement that they were no longer willing to censor searches in China and that they may pull out of China altogether. On March 23, 2010 Google began redirecting all google.cn searches to Google.com.hk – its Hong Kong

search engine. Although Hong Kong is part of China, it is not subject to China's restrictive censorship laws.

On March 30. 2010 China banned access to all Google sites. Apparently any attempt to access a Google site resulted in a DNS error. However the ban lasted just 24 hours. The redirection to the uncensored Hong Kong site then continued.

Google was due to have its licence to operate in China renewed on June 30, 2010. When the day came, Google restored its China search engine complete with censorship. They did however include a link to their Hong Kong site. Both Google and the Chinese Government have claimed victory. Complete information is not currently (2011) available, however it would appear Google backed down under pressure that it would be thrown out of China.



GROUP TASK Research
China is not the only country to censor the internet. Research and summarise internet censorship occurring in different countries.

Napster versus Metallica (and RIAA)

Napster was a Peer-to-Peer (P2P) file sharing internet service that allowed people to share music without having to purchase an original CD. Napster's application, once downloaded allowed users to access and download songs that are stored on other users hard drives. Such was the success of Napster that in the first few months, it was estimated that more than 85 million people around the world used the service.

In 1999, the Recording Industry Association of America (RIAA) filed a lawsuit on behalf of five major US record labels alleging Napster violated both Federal and State copyright laws and stated that *"Napster knowingly and wilfully set out to build a business based on copyright infringement on an unprecedented scale."* and in 2000 the heavy metal band Metallica files a similar lawsuit against Napster.

Napster during their defence argued that they are simply an Internet Service Provider but are also protected by the 1992 Home Recording Act which allows the general public to make a copy of music for personal use. The judge disagreed saying the service did violate copyright and issued an injunction to have the site shutdown. The US government also states that Napster's service falls outside of the definitions of the Home Recording Act.

Just hours before the site is due to be shut down the US Circuit Court of Appeals issues a stay of the injunction until Napster's full appeal can be heard. During the appeal Napster's defence team insist that it is the end-users that are in violation of copyright and not Napster, saying the company only provides the means for users to trade music, which in itself, is not an infringement of copyright.

In September of 2001, Napster reaches a partial settlement of the lawsuits filed against it and agrees to pay US$26 million to some music publishers and songwriters. Napster also states that it intends to become a subscription-based service with a percentage of royalties being distributed to the copyright owners of the music.

Agreement with the major record labels could not be reached however and in 2002 Napster filed for Chapter 11 bankruptcy.



GROUP TASK Research
Peer-to-Peer (P2P) software is still in widespread use. Research to determine its operation and legal status.

CHAPTER 1 REVIEW

1. Software developers need to achieve:
   (A) a respect for the marketing departments in software companies.
   (B) a balance between their own rights and their responsibilities to their clients.
   (C) a relationship between their copyright and royalty rights.
   (D) strong personal relationships with clients.

2. Which of the following classifications of software cannot be copied and distributed?
   (A) Commercial
   (B) Open Source
   (C) Shareware
   (D) Public Domain

3. Responsible software developers will ensure:
   (A) their products do not influence the execution of other installed applications.
   (B) they make copies of competitor's products.
   (C) they make changes to any modules of code they use from other products.
   (D) All of the above.

4. Owning a software licence usually means:
   (A) you can freely copy, modify and distribute the product.
   (B) you can use the software and make one copy for archival purposes.
   (C) there is no need to enter into a contract with the copyright owner.
   (D) the product can be installed on any machines that you own.

5. Security measures to combat software piracy include all the following EXCEPT?
   (A) encrytion
   (B) registration coes
   (C) back-to-base authentication
   (D) heat sensitive ink

6. A set of standards to which professional software developers agree is called:
   (A) a Code of Conduct.
   (B) Quality Assurance.
   (C) Beta testing.
   (D) a Contract.

7. Copying software and then selling it under the guise of being the original is known as:
   (A) plagiarism.
   (B) piracy.
   (C) counterfeiting.
   (D) theft.

8. Operating system developers:
   (A) must maintain their market share at all costs.
   (B) should introduce completely new products as new technologies arrive.
   (C) have a responsibility to ensure their products are well documented and reliable, as software must interface with these products.
   (D) cannot be expected to consult with application developers when designing new products.

9. When a runtime error occurs:
   (A) the program should display a meaningful error message to assist in isolating the problem.
   (B) well written applications will do their best to save any data before execution halts.
   (C) error routines should attempt to isolate and correct the problem so that execution can continue.
   (D) All of the above.

10. Support for software products is the responsibility of software developers. Software support includes all of the following EXCEPT:
    (A) Isolating problems.
    (B) Creating bug fixes.
    (C) Updating.
    (D) Training.

11. Define the terms Intellectual property, copyright, and software licence agreement. Explain the relationship between each of these terms in regard to software products.

12. As a software developer, you have ethical responsibilities to uphold for the good of the industry. Describe these ethical responsibilities. Include areas related to the quality and reliability of the product, upholding the intellectual property rights of others, and software marketing in your response.

13. A school wishes to purchase a multimedia CD-ROM product. The teacher purchases one copy and then installs the product on the school's network where it can be used by anyone with network access. Discuss the legality of this situation in terms of copyright.

14. A group of software developers are creating products to automate tasks within the Real Estate industry. These software developers form an association to help maintain standards within their particular section of the market. Design a code of conduct for this association.

15. A software developer creates a new product. This product accepts object code as its input. The product analyses this code and outputs a listing similar to a high-level language. The software developer decides to market the product as 'Shareware'. Discuss the possible social and ethical issues involved in this scenario in terms of the developer and the users.

In this chapter you will learn to:

- compare and determine the most appropriate software development approach for a given scenario
- communicate understanding of a commercial system studied using a case study approach by:
  - identifying the approaches used
  - discussing the appropriateness of the approaches used
  - describing how the various personnel contribute to the overall development
  - critically evaluating how social and ethical issues were addressed
  - evaluating how effectively the new system met the needs of the user
- make informed comment on current trends in software development

Which will make you more able to:

- differentiate between various methods used to construct software solutions
- explain the interrelationship between emerging technologies and software development
- identify and evaluate legal, social and ethical issues in a number of contexts
- apply appropriate development methods to solve software problems
- apply project management techniques to maximise the productivity of the software development
- create and justify the need for the various types of documentation required for a software solution
- select and apply appropriate software to facilitate the design and development of software solutions
- assess the skills required in the software development cycle
- communicate the processes involved in a software solution to an inexperienced user.

In this chapter you will learn about software development approaches

Approaches used in commercial systems, including:

- Structured approach
- Agile approach
- Prototyping
- RAD (Rapid Applications Development)
- End user approach
- combinations of any of the above

Use of Computer Aided Software Engineering (CASE) tools and their application in large systems development, including:

- software version control
- test data generation
- production of documentation
- production of code

Methods of installation of new or updated systems

- direct cut-over
- parallel
- phased
- pilot

Employment trends in software development, for example:

- outsourcing
- contract programmers

Trends in software development

- changing nature of the environment in which developers work while creating software solution
- changing nature of applications
  - web-based software
  - learning objects
  - widgets
  - apps and applets
  - Web 2.0 tools
  - cloud computing
  - mobile phone technology
  - collaborative environments

2

APPLICATION OF SOFTWARE DEVELOPMENT APPROACHES

APPROACHES USED IN COMMERCIAL SYSTEMS

When new software products are first considered for development the initial planning is vital. Software development companies need to decide on the most appropriate software development approach, the most suitable method of installation, as well as what resources they will require to complete the task in the most efficient manner.

A software development approach is a model of the general technique used to produce a software product. All software development approaches use the five steps of the software development cycle.

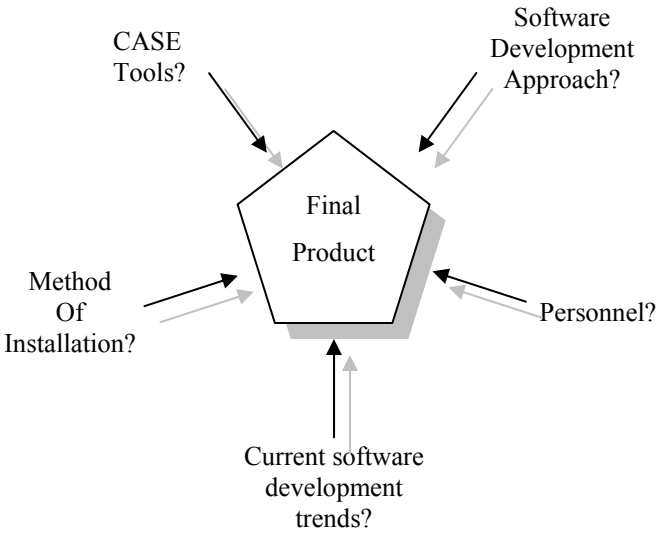


Fig 2.1
Initial decisions that must be made before development of a new software product.

These steps are as follows:

1. Defining and understanding the problem.
2. Planning and design.
3. Implementation.
4. Testing and evaluating.
5. Maintenance.

It is the timing and frequency of these steps that changes for different approaches. For example, in the structured approach each step must be completed before you progress to the next step. However, in the agile and prototyping approach these five steps will be continually revisited, as the product is progressively refined.

The approach or methodology chosen is influenced by a number of factors, including:

- The size of the product
- The nature of the product
- The skills of the development personnel
- The detail of the requirements
- The finances available to fund the product development.

Often a combination of approaches will be appropriate. However, it is worthwhile examining five common software development approaches and analysing the strengths and weaknesses of each.

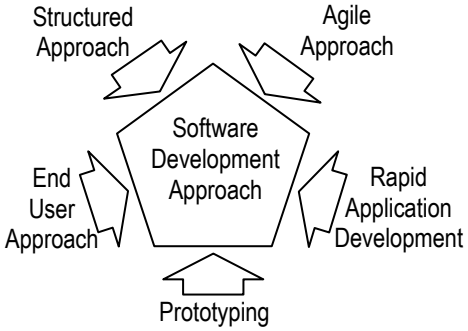


Fig 2.2
Five common software development approaches, which can be used in combination.

THE STRUCTURED APPROACH

This approach, as the name suggests, involves very structured, step-by-step stages. Each stage of the software development cycle must be completed before progressing to the next step. The structured approach closely follows the development processes performed for most products. That is, the stages involved in development must occur in an orderly and sequential fashion. In other words, you cannot implement a solution until you consider potential solutions, and potential solutions cannot be envisaged until you have the problem thoroughly defined and understood.

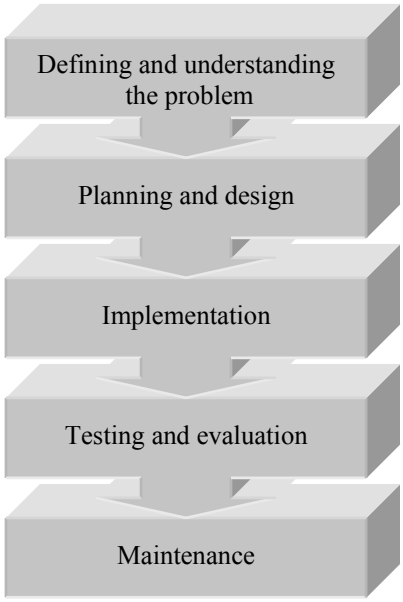


Fig 2.3
The structured (or waterfall) approach. Each step is completed before the next step commences.

This is the traditional approach to software development. Often this method is known as the 'waterfall method', as you start at the top and work progressively down to the bottom. This approach mirrors the structured, top-down approach used in the design of algorithms for procedural programs. There is a distinct start and finish and each step is completed in a predetermined, precise order. Because of the structured nature of this method, it is particularly suited to software development where all the requirements of the product are, or can be, defined and understood before any planning and design commences.



Consider the following:

A company produces and sells a product that needs to interface with various light sensor units produced by differing manufacturers. A driver needs to be developed for each new light sensor unit that is released onto the market.



Driver
In this context it provides the interface between a hardware device and an application.

This scenario is particularly suited to a structured approach. The requirements for the new drivers are known precisely. The software product already exists so the input requirements are already known. The hardware manufacturer will also know the precise format and nature of the output from the light sensor unit that they produce. A solution therefore can be planned, designed, implemented, tested and evaluated without the risk that the requirements will change during the software development cycle.

The structured approach to software development has many advantages for custom systems that are being built for large-scale implementation where the specifications can be defined precisely. Often these types of applications are specific to a particular industry or product and perform quite specific sets of tasks.



Consider the following:

A new model of DVD recorder is under development by a manufacturer. They contract a software development company to produce the necessary software to operate the DVD recorder.

This software product will be used exclusively for controlling the new model of video recorder. The hardware specifications of the DVD recorder are known. A structured approach to the development of this software would be the most appropriate choice. Some changes to the design of the user interface via a remote control may need to be made, but on the whole the specifications for the system are well defined and understood. Most of the processing in the software would be involved in actually controlling the operation of the machine.



Validation
The process of substantiating the requirements of a system. In this case, ensuring that particular requirements have in fact been met.

Other software development approaches have primarily been introduced to counteract the problem of having to create a complete set of requirements and specifications prior to the commencement of the design stage. The structured approach assumes that it is possible to create a complete specification, when in practice this is not normally the case. It is much more likely that many requirements and specifications will not be realised until development of the software is underway. In fact, often it is impossible to generate all the requirements until testing of some sort has validated parts of the system. The bulk of errors encountered during the software development cycle occur during the 'defining and understanding the problem' stage. Studies have shown that at least 56% of errors occur during this initial stage. These studies also indicate that problems occurring at this stage become increasingly expensive to correct as the software development cycle progresses.

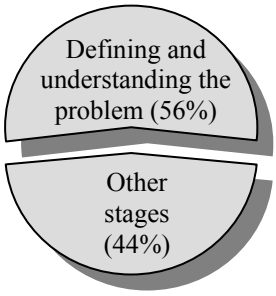


Fig 2.4
Source of errors during the software development cycle.

An error due to a problem or oversight in the initial 'defining and understanding the problem' stage will cost 5 times as much to correct at the 'planning and design' stage, 10 times as much to correct at the 'implementation' stage, 40 times as much to correct at the 'testing and evaluation' stage and a massive 200 times more if left unresolved until the 'maintenance' stage. If the structured approach is to be used, then clearly developers must be extremely careful to set the requirements and specifications very accurately.

Stage	Relative cost of repair
Defining and understanding	1
Planning and design	5
Implementation	10
Testing and evaluation	40
Maintenance	200

Fig 2.5
Relative cost of error correction for errors originating at the initial 'defining and understanding the problem' stage.



Consider the following:

A chain of beauty salons located in five adjoining suburbs of a major city wishes to be able to access client records from all stores. Each of the stores already has permanent Internet access. Personnel employed by the chain of salons do not have the IT skill required to write a system, so they contract the services of a software development company.

The scenario outlined above is very common. Often companies have no technology expertise in house, so they outsource the development to a company that does.



GROUP TASK Discussion
Why do you think the beauty salon chain might want to share customer records amongst its suburban branches?



GROUP TASK Discussion
Would a structured approach to the development of this software solution be feasible? Explain your reasoning.



Consider the following:

A pool repair company specialises in replacing the liner in vinyl pools. They are also equipped to replace filters, pumps and the associated plumbing. They purchase their materials from one particular supplier. The pool repair business has been operating for thirty years. They have developed a unique but extremely organised paper-based quoting and invoicing system. They now wish to computerise not only the quoting and invoicing side of the system, but also the ordering of materials from the supplier, along with automatically produced monthly reports.



GROUP TASK Discussion
Is this project suitable for a structured software development approach? Explain and discuss.



GROUP TASK Discussion
Discuss how the requirements of this system can best be determined so that the old system can be computerised without loss of efficiency.

AGILE APPROACH



Agile
Quick thinking, co-ordinated, active and lean. Adapts well to changing situations.

Agile development methods have emerged in response to the "adhoc" reality of many software development projects. They place emphasis on the team developing the system rather than following predefined structured development processes. Agile methods remove the need for detailed requirements and complex design documentation. Rather they encourage cooperation and team work. Agile methods are particularly well suited to web-based software development and other software applications that are modified regularly such that they evolve and are updated over time.

Typically quite small teams of developers are used. It would be unusual for an agile team to have more than about half a dozen members. It is preferable for one team member to be a knowledgeable and experienced user. Small teams are better able to share ideas and work on solutions together. Larger teams tend to break into smaller groups – for agile methods to be a success everyone must be an equal member with a clear shared purpose. Often the team members are multi-skilled so all are actively contributing to all development activities.

Typical characteristics of an agile software development approach include:

- Speed of getting a working solution to market or to users. Basic functionality is included initially so operational software can be released as soon as possible.
- Interaction within the team and users which allows the solution to be selectively refined throughout the development process.
- Working versions of the software are regularly delivered. Each version adds the next most important functions.
- Responds well to changing specifications. Specifications are encouraged to change and evolve.
- Development team and clients collaborate closely througout development. Often a client representative or user is part of the team. The needs and ongoing feedback of the client and users drives the direction of the development.

Let us work through the typical sequence of activities occuring during agile development (refer *Fig 2.6*). Initially the general nature of the problem is determined and the development team is formed. The team first meet to create a basic plan and a general design for the software – only minimal detail at this stage, just enough to get started. The basic idea is to only plan, design and document details as they're actually needed. Often a simple whiteboard is used to sketch out the general design. The team then gets straight to the task of creating an initial solution. As this occurs they informally consult and negotiate with each other. The user team member is always present to answer questions, make suggestions and generally ensure the solution will be workable in practice.

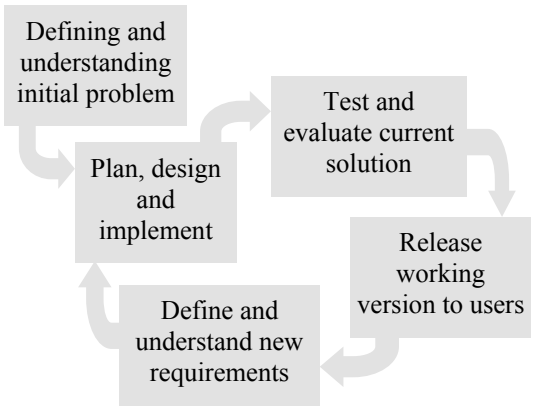


Fig 2.6
The software development cycle for the agile approach.

Once an initial, yet simplified, solution is implemented it is immediately tested, evaluated and then released for use. This means a working version of the software is actually being used by real users – usually the client but it could be a sample of users or even globally to all users via the web. The users see exactly what has been achieved, can provide feedback and make suggestions about further additions. In effect we have entered a new mini "defining and understanding the problem" phase. The team again meets informally to discuss the next part of the design. The design incorporates feedback from users together with their own ideas. They then go straight to work coding this next part of the design. The solution is again thoroughly tested and evaluated before being released. This process repeats many times with each iteration implementing further functionality and detail. Typically a single iteration

takes just weeks or even days. Each design meeting is short, maybe just an hour or so, whilst coding and testing consumes the majority of the development time.

In many Agile approaches the focus is on the user and what they want the software to actually do. Often "user stories" are created by each different type of user (refer *Fig 2.7*). These user stories are prioritised and each becomes the focus of a development iteration. For example in *Fig 2.7* customers want to be able to look up their orders to see if they have been shipped. This is a user story and hence implementing this user story in the software will directly meet a customer need. Remember a representative user is part of the team so they are readily available to assess how their user story is being implemented in code as it is actually being designed and developed. Each user story involves one or more actors (often users, such as customers or employees) and is refined using suitable system models. Currently UML (Unified Modelling Language) is the industry standard for modelling. The UML includes a variety of diagrams (including DFDs) and notations for modelling different aspects of software systems. For many teams user stories are simply written on slips of paper or on a whiteboard, whilst other teams may utilise software tools such as the example reproduced in *Fig 2.7*. During design and development new user stories will emerge, some will be removed, others will be reprioritised and others maybe split into multiple user stories or combined with other user stories.

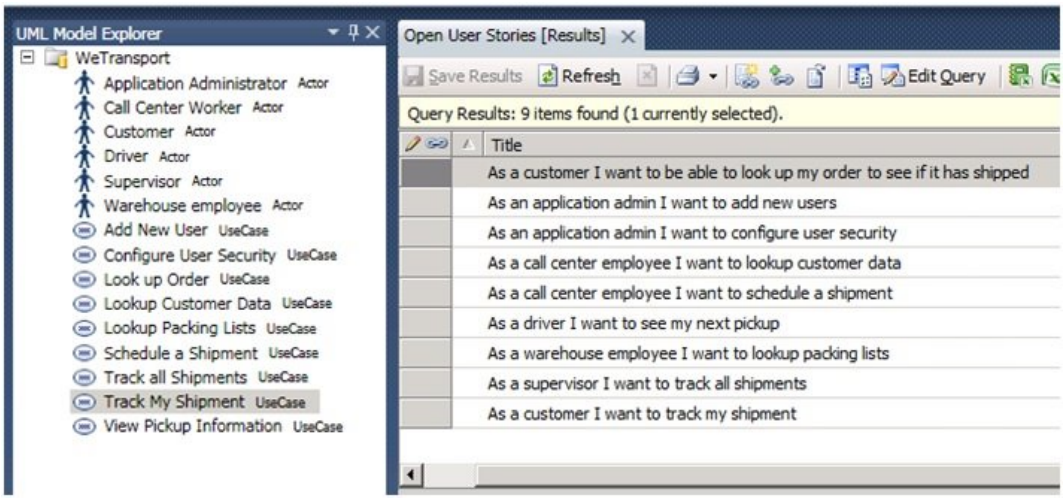


Fig 2.7
Sample user stories within Visual Studio.



GROUP TASK Research
Different diagrams, models and notations within the UML (Unified Modelling Language) are integral to many Agile approaches. Research the different diagrams, models and notations that form part of the UML.

When developing software it's all the miniscule details that combine to form the total solution. Agile methods are a response to the reality that intricate details are difficult to specify accurately in advance. Each part of a software solution relies heavily on many other related parts. Until the related parts exist, it is wasteful to continue designing. Much of the design will prove unworkable and will need to be redesigned or significantly altered. Compare this to the traditional structured approach where specific and intricate detail is created well in advance.

One significant issue with agile methods is how to construct agreements when outsourcing the development. Traditionally a strict set of detailed requirements, together with the total cost and time for completion is negotiated. When using agile

methods no detailed specifications exist – they emerge and change during development. A common solution to this dilemma is to fix the budget and time and allow the specifications to change. Once the budget and time is exhausted then the current solution becomes the final solution. To enter into such agreements requires significant trust to be established between the client and developer. The client stands to gain, as they are heavily involved throughout the development process and hence are more likely to receive a final product that better meets their actual and current requirements.



Consider the following:

The following manifesto was written in 2001 by 17 diverse software development experts during the formation of the Software Development Alliance, better known as the Agile Alliance (www.agilealliance.org). The manifesto defines four core values to encourage better ways of developing software.

The Manifesto for Agile Software Development

We are uncovering better ways of developing software by doing it and helping others do it.

Through this work we have come to value:

- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan

That is, while there is value in the items on the right, we value the items on the left more.



GROUP TASK Research

The Agile Alliance (www.agilealliance.org) provides a variety of resources for those using or intending to use an Agile approach to software development. Research the initiatives and resources provided by the Agile Alliance. In particular, ensure you read their "Twelve Principles of Agile Software" which expand upon the above manifesto.



GROUP TASK Research

There are a variety of different versions of the agile approach. For example, Scrum, Extreme Programming, Crystal, and Lean software development. Research a variety of agile methods and summarise their major features.



GROUP TASK Discussion

Many agile methods include a test driven development (TDD) methodology. Essentially tests are always coded before writing any actual code. The tests will of course fail before the actual code is written and will pass once the code is correct. Numerous small tests are added to the test suite as development progresses and all these tests are rerun each time new code is added to the project. Research and discuss advantages of TDD.

PROTOTYPING

Prototyping, as a software development approach, predates agile approaches. It lies between the traditional structured approach and newer agile approaches. Prototyping lends itself to intense interaction between customers and software developers. However the prototypes are created to progressively refine user requirements. Like agile approaches, the prototyping approach is iterative. Throughout the development process the customer successively validates the product. Each successive prototype better meets the original and realised requirements for the final product. In some cases the final prototype will be further developed into the final product, however this is not essential. Often the prototypes are simply interactive models of the user interface.

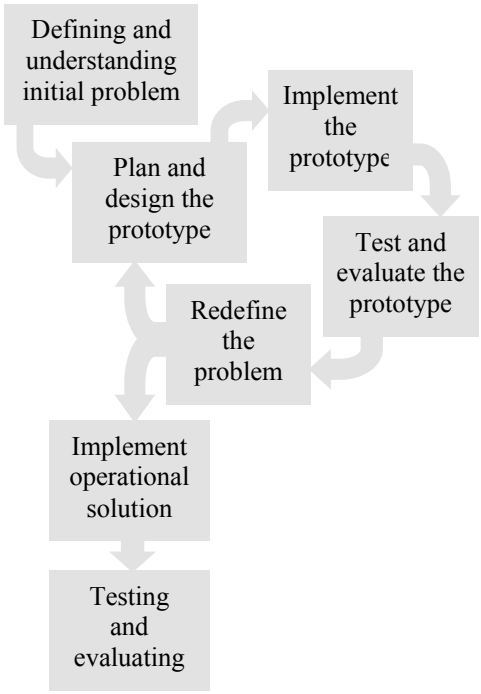


Fig 2.8
The program development cycle for the prototyping approach.

Prototyping (much like more recent agile approaches) came about as a recognised software development approach when it was acknowledged that many requirements for software products were not fully known, and in many cases could not be known, until the product's design had commenced. The traditional structured (sometimes known as 'waterfall') approach was unable to react to changing requirements once the planning and design phase had commenced. The prototyping model acknowledges that the requirements for a system are collected in an evolutionary fashion. Knowledge of requirements is never complete but rather evolves over time, as new, previously unknown requirements start to emerge, existing requirements become modified, and some already identified requirements may be discarded.



Prototype
An enactable model or mock-up of a software system that enables evaluation of features and functions in an operational scenario.

There are two major types of prototypes that can be utilised by a software developer:

- Concept Prototypes
- Evolutionary Prototypes

Concept Prototypes

This prototype is never meant to be the final product. The aim is to promote discussion and further enhance the developer's knowledge of the customer's requirements. A number of prototypes may be presented to the customer simultaneously. Each of these prototypes may be the result of different approaches to the solution of

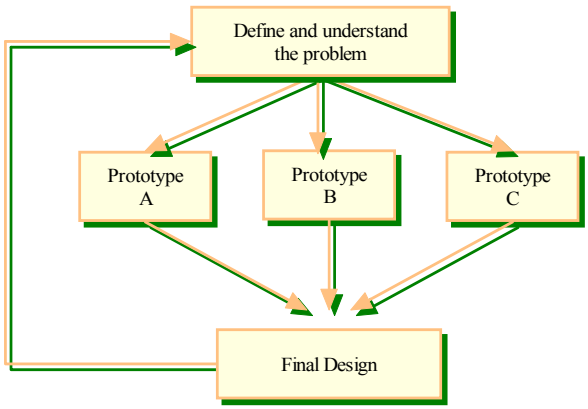


Fig 2.9
The Concept Prototyping Approach.

the same problem, or they may represent solutions to differing parts of the same problem. Often concept prototypes are referred to as rapid, throw away, experimental or exploratory prototypes. Concept prototype is the more general use of the term 'prototype', where a concept is presented to assist in developing and modifying requirements and specifications.

Evolutionary Prototypes

This kind of prototype is intended to be the forerunner to the final product. In other words, the initial prototype will eventually become the final product. Each successive generation of prototypes is an improvement on its predecessor. Eventually a level of functionality is reached where the prototype becomes the final product. With almost all non-software products it is not possible for the prototype to evolve into the final product. For example, a scale model of a vehicle design is clearly not a functional prototype, therefore it can never evolve into the finished product. Evolutionary prototyping works extremely well for software solutions, as the manufacturing and distribution costs are minimal compared to most other products.

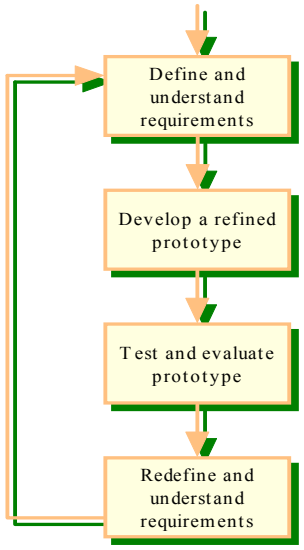


Fig 2.10
The Evolutionary Prototyping Approach.

Many software developers expand this evolutionary prototyping approach even further. In essence, they consider each new version of their product to be an evolutionary prototype. This means that the traditional maintenance stage becomes an integral part of 'defining and understanding the problem'. In other words, the solution is never found but is continually sought after in successive versions and updates. This method acknowledges that software products are never perfect, but that the developers of quality software need to continually strive for this perfection.

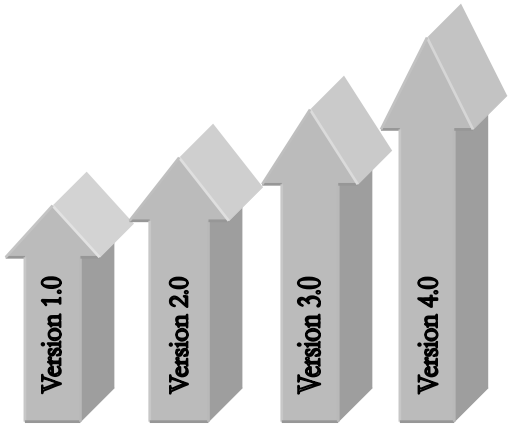


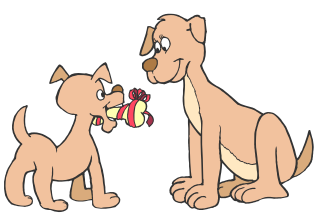
Fig 2.11
Evolutionary prototyping can expand to encompass new versions.

This method acknowledges that maintenance and support staffs are the people who are continually involved in communication with the users of the software, and hence their role in defining and understanding the problems and expectations of a product should be highly valued.



Consider the following:

A web site design company has been contracted to develop a site to advertise and sell a new range of pet care products. They are presented with a brief that includes photographs, descriptions and prices for each pet care product, together with a requirement that the products can be purchased online.





GROUP TASK Discussion
Which form of prototyping would be most appropriate *initially* for the above scenario? Explain your reasoning.



GROUP TASK Discussion
Let us assume that the pet care business approves an initial design. How should the web site design company progress if they wish to use an evolutionary prototyping approach?



Consider the following:

A multinational software development company, known as 'FoogleSoft', has developed, and introduced onto the personal computer market, a new operating system called 'GlassPane'. The product is a success and captures 30% of the market. As with any new product there are many bugs and many possible improvements that could be made to the software.



GROUP TASK Discussion
Discuss a software development strategy suitable for this company that would allow for successive upgrades of the 'GlassPane' product. Justify your choice of software development strategy.



GROUP TASK Discussion
After 5 upgrades of the original 'GlassPane' operating system, FoogleSoft decides to investigate the possibility of a significant change to the main user interface for the product. Discuss a possible strategy that could be used to investigate the feasibility of different user interface designs.

RAD (RAPID APPLICATION DEVELOPMENT)

Rapid Application Development, as the name suggests, is a software development approach that allows usable systems to be built within a small amount of time. The RAD approach was introduced to overcome the long delays experienced by customers when more conventional structured methods are used. Often software development, using traditional means, could take so long that the requirements for the system had fundamentally changed by the time the new system was ready for use. RAD aims to reduce development time so that the majority of requirements can be met and implemented as quickly as possible. There are disadvantages to RAD: reduced quality of the final product is the most likely outcome of selecting a RAD approach. The quality compromises are likely to be in terms of a less useable, less fully featured or less efficient product, rather than a larger number of errors.



4GL
Fourth generation language. The emphasis is on what has to be done rather than the detail of how it is accomplished.

RAD has only become possible as a result of the development of Fourth Generation Languages (4GLs). These languages enable a 'visual' production of user interfaces and the integration of reusable components into new products with very little hard programming. Many applications can be developed with very little coding taking place at all.

For the RAD approach to be successful, it is necessary for the team developing the product to work very closely. The team should include multi-skilled developers together with full-time users of the system. Continual feedback from the users in the team will be the main method of determining which requirements are critical and which are secondary (and hence may be discarded). Remember, the aim of the RAD approach is to get a working product up and operational as fast as possible, even if some secondary requirements are not met. An important task of the team is determining which of the requirements must be met and which can be ignored.

Requirements for the use of the RAD approach:

Personnel

- Teams comprising developers and users.
- Developers need to be analysts, designers and programmers rolled into one.
- Users need to be fully involved, full-time users who have a stake in the requirements being realised.

Tools

- 'Visual' development, 4GL environments.
- Multiple languages can be used.
- Standard APIs.
- CASE tools.
- Prototype and simulation tools.



API
Application Programmers Interface. A set of functions and procedures built into the operating system that can be used by programmers.

RAD tends to work when:

- The application will be run on standalone machines.
- Major use can be made of existing modules of tried and tested code.
- Performance of the system is not critical.
- Product distribution is narrow, often for a specific client.
- Reliability of the final system is not critical.

RAD tends to fail when:

- Applications must interface with other software.
- Few existing modules can be used during development.
- Optimal performance is a critical system requirement.
- Product distribution is to a broad audience.
- Reliability is a critical system requirement.



Consider the following two scenarios:

- A new type of CPU has been designed and is now in production. A software development company has been contracted to develop a new operating system specifically for this CPU.
- A computer game company is developing a new 3D graphics game. The game contains complex animation sequences that require maximum processing speeds to redraw the screen.



GROUP TASK Discussion
Is RAD a suitable approach for the development of these new systems? Explain your response in relation to each scenario.

Often the RAD approach uses a prototyping methodology to achieve its purposes. The prototypes are used in team meetings to help clarify requirements for the new system. As with the prototyping approach, often the prototype evolves into the final product. This is known as 'Rapid Prototyping', a fast prototyping approach.

Advantages and disadvantages of RAD include:

Advantages

- Buying modules may save money compared with building them from scratch
- Greatly reduced manual coding
- Possibly fewer code errors as code is produced by CASE tools
- Shorter software development cycle resulting in reduced costs
- Standardised look and feel because of the use of APIs and other reusable components

Disadvantages

- Buying modules may be more expensive than building them from scratch
- Other software may be required to be installed on customers' machines
- Less efficient code because it is not hand crafted for the particular job
- Reliance on third party modules may sacrifice needed functionality
- Reliance on third party modules may add unneeded functionality
- Using third party components may create legal problems

Characteristics of the RAD approach include:

Lack of formal stages

Design and development of products progress informally. The needs of individual clients and the individual product determine the manner in which the product is developed. A free flow of information and ideas is crucial to success. As the software is produced by combining and customising other products the necessity for formal models, data structures and algorithms is greatly reduced. This unstructured approach works well when a small number of developers and end users collaborate closely during development. As the scope of the project is limited each team member is able to have an intimate knowledge of the entire system.

Use of existing routines

Wherever possible existing code routines are used to reduce the development effort. Extensive use is made of the application program interface (API) included as part of most operating systems. For example, API calls could be used to access standard dialogue windows for opening files, printing and searching. The final application may utilise the services of various applications and existing code modules to complete its task.

Appropriate applications to develop RAD solutions

Many RAD applications are developed using the scripting language built into larger software packages. For instance Microsoft Office includes Visual Basic for Applications (VBA) which is used by programmers to build custom applications. VBA uses almost identical syntax as Visual Basic Version 6.0 therefore code and routines developed and tested in Visual Basic 6.0 can be used to customise Office applications. In this case the final product is known as a Customised Off-The-Shelf or more simply COTS package. Microsoft Office being the Off-The-Shelf package.

A graphical integrated development environment (IDE) greatly improves the productivity of most RAD programmers. The ability to graphically draw screens and screen elements using drag and drop simplifies the process. Many IDEs also include wizards which automate the coding of screens. Often the default behaviour of screen elements built using a wizard requires minimal or no coding.

Communication between developer and client

The developer(s) and client(s) should work together as a team. Often a single developer will use RAD to create an application for use by a specific client. Continual communication between developers and end users to determine and refine requirements during development is vital. The software tools used for development allow the programmers to alter the product in a short time period. Consequently, responding to new and changing requirements occurs quickly and with the full knowledge of all involved.

Short time period

The main aim of using the RAD approach is to shorten development times. Often requirements will be compromised in the interests of expediency. Research has shown that requirements often change significantly whilst projects are being developed. The RAD approach addresses this issue through the use of existing solutions and modules together with integrated development environments.

Small-scale project

RAD is particularly suited to small-scale projects. Often, the client will be a small business and the developer an individual. This does not mean the applications developed using RAD are unsuited to wide distribution; rather they focus on achieving a limited set of requirements. For example, the software behind many multimedia CD-ROM and DVD titles is produced using RAD tools and techniques. These products are designed to execute on standalone machines and utilise the resources of various multimedia players. A single developer creates the software for the multimedia company who creates the graphics, video and other data. The product is then distributed through appropriate retail outlets.

Low budget

Many software products developed today using a RAD approach would not have been feasible prior to the existence of modern 4GLs and customisable applications. It was just not economically viable. RAD allows quality applications to be developed at low cost. The widespread reuse of components is common in most industries but is relatively new to software development. For example, tyres for cars can be used on various vehicle models, steel is used in the manufacture of bridges, nails, fencing, etc. The RAD approach to software development encourages the reuse of software components or modules resulting in subsequent cost savings. Low budget software applications are particularly suited to development using RAD techniques.



Consider the following:

A utility has been built using a RAD approach. This utility is designed to search all the directories (or folders) on a hard drive and categorise the files it finds according to criteria entered by the user. This utility makes use of many MS-Windows API calls to find the files and to generate the windows used to display the results.



GROUP TASK Discussion
A new version of MS-Windows is introduced that includes various changes to the look and functionality of individual windows. How could this affect the operation of the utility described above? Consider both positive and negative effects.



GROUP TASK Discussion
The software development company is approached by Apple Computer to develop a similar utility to be included with the next version release of their operating system. This is very exciting for the software development company and they wish to ensure they can comply with Apple's request. Can the utility be easily modified for this new Apple environment? Explain your answer.

END USER DEVELOPMENT



End User
The final person or persons who operate a software package.

The End User approach to software development has become increasingly popular, as application packages have been released onto the market that can be customised by users with limited software development expertise. This End User approach involves businesses creating their own products by customising other applications using wizards and other automatic code generating devices.

End User development of software has many advantages for small businesses and home users as they would not otherwise be able to afford customised solutions. They are also able to continuously revise and upgrade these solutions as requirements change. It gives small businesses the ability to automate their functions at low cost.

There are however a number of disadvantages of End User development. The final product will need to be a compromise between the requirements of the business and the functionality of the application being customised. Because of the general nature of the original application, the resulting product will not be as efficient as a similar product written from scratch. This means performance can often be less than satisfactory. Errors often emerge as a result of lack of knowledge on the part of the user-developer, resulting in a lower quality product.

Characteristics of the end user approach include:

Use of standard software packages

Most end user developed products are produced using packages already owned by the user. The experience as a user of the package gives them sufficient knowledge to customise the product to suit their particular needs. In this regard end-user development typically uses the RAD like or prototyping like approach to develop a COTS (Customised Off-The-Shelf) package. For specific development problems the developer/user will often refer to online resources such as, news and user groups devoted to customising the base software package.

End user as the developer and maintainer

The end user has a need or idea for a specific software solution. They develop the software using tools with which they are familiar. The solution focuses on solving the

problem with little regard to testing or usability issues. As the developer is the user then clearly they will understand how the software operates without the need for training material or help systems. Clearly there are no communication issues when the developer is the user.

Small budget and/or short time period for development

Often the development costs are nil. The end user's time and perhaps the cost of the parent application being the only significant costs. The low cost is an incentive for many users to develop their own software solutions.

Development is often completed within hours, at the very most a few days. The size and complexity of these projects is such that they can be accomplished within a small amount of time. Often the software is not central to the user's job, rather it is used as a tool to streamline or improve the efficiency of some aspect of their job.

Lack of formal stages

Development is unstructured, informal and is often undertaken with little initial planning. The user-developer is often learning the package as they develop their product. No formal attempt is made to document the solution. As the software is normally for the exclusive use of the user-developer the need for documentation is negligible.



Consider the following:

A salesman wishes to maintain a record of all the potential customers he encounters together with their individual product preferences. He wishes to be able to select certain customers according to entered criteria so that those customers may be faxed newsletters and product details directly from his computer.



GROUP TASK Discussion
Let us assume the salesman has a good knowledge of computerised database management systems. Is it feasible for the salesman to create this system himself? Make some suggestions in regard to the products he needs to purchase to complete the project.



GROUP TASK Discussion
List advantages and disadvantages for the salesman as a result of developing this system himself.



Consider the following:

A corner store wishes to automate its stock control and ordering functions. As its finances are restricted, it decides to use 'Datamaker Pro' as the basis of its new product. The storeowners, Bob and Susan, already own a licence to use 'Datamaker Pro'. They will attempt to customise 'Datamaker Pro' to suit their requirements.



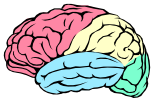
GROUP TASK Discussion
Bob and Susan have no real database experience. Do you think they would be better served by purchasing a package designed for stock control in stores, even though these packages are costly? Debate.



GROUP TASK Discussion
Bob and Susan go ahead and develop their product. Soon after, a friend of Bob and Susan's, who owns the local Newsagency, wishes to implement a similar system for their stock control. Bob lends the Newsagency his 'Datamaker Pro' CD-ROM together with a copy of his stock control product. Discuss the social and ethical issues associated with this scenario.

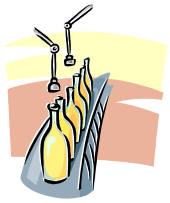
COMBINATIONS OF SOFTWARE DEVELOPMENT APPROACHES

Often development of a software product will be best accomplished with a combination of approaches. Perhaps particular modules will be developed using a structured approach, and others using a prototyping approach. The nature of each particular problem needs to be assessed so the most appropriate approach or approaches can be selected.



Consider each of the following scenarios:

- A soft drink bottler wishes to computerise its bottling production line. They have purchased the physical bottling hardware and sensors, but because of the specialised nature of their particular bottling plant, no software is available. They wish to have the necessary software developed to control their bottling plant.



- The weather forecasting bureau wishes to provide hourly forecasts over the Internet for viewing by the general public. At present their forecasts are stored on their internal network in a non-user friendly format that can only be understood by trained meteorologists.



- The Australian Taxation Office wishes to produce a knowledge base of all the rules and regulations applicable to the taxation system. This knowledge base will be used by taxation office personnel to allow them to better answer questions from the general public.





GROUP TASK Discussion
For each of the above scenarios, describe the most appropriate software development approach or approaches. Justify your choices.



HSC style question:

A doctor's surgery employs two doctors, a nurse and a secretary. They have employed a programmer to develop a software application to manage appointments and patient records. The programmer has decided to use a combination of the *prototyping* approach and *rapid application development* (RAD) approach to develop the software.

(a) Define the *prototyping* approach to software development.

(b) Define the *rapid application development* approach to software development.

(c) Explain likely reasons why the programmer would choose to use a combination of *prototyping* and *RAD* approaches to develop the software for the doctor's surgery.

(d) The programmer intends to sell the completed program to other doctors' surgeries. Outline the issues involved.

Suggested Solution

(a) The prototyping approach focuses on the interfaces that will be used in the final product, and the links between them to demonstrate what the proposed product will look like from the users perspective. There is no development of logic or error trapping as will be required in the final product, as this approach is used to quickly develop the required interfaces using drag and drop facilities. This allows immediate feedback from the client, facilitating inclusion of their requirements and suggestions into the overall look and feel of the prototype. This ensures a full understanding of the requirements of the new system by both developer and client before the product is developed, as well as a better involvement in and ownership of the product by the client from the start of the project.

(b) The RAD approach allows the developer to make use of predefined logic routines available in a 4GL software package to develop a full application much quicker than having to write all of the required code in a 3GL language in a much more formalised manner. The product developed using the RAD approach is developed in a quicker timeframe on a smaller budget, usually by the developer working directly with the client to ensure their needs are met. This approach works best when the required solution utilises the generic logic already available in the RAD software package.

(c) A RAD approach can develop a robust solution in a fast time frame and on a limited budget, but to meet the needs of this particular doctor's surgery, it is essential that the solution is appropriate for them. Thus, both developer and client need to go through the prototyping stage first to ensure that the screens they will be using in the final product include the required data, presented in an appropriate format and using a colour scheme that is appropriate for their use. They must both also be comfortable with the functionality offered by the prototype interface. Once both developer and client are happy with the proposed solution, then the developer can go away and develop the solution using a RAD approach so that it looks the same as the agreed demonstrated prototype.

(d) The developer must not include any code in the solution that they have not developed themselves or that was not available in the RAD application being used to develop the product. They cannot 'borrow' any routines from another programmer or from the Internet without the permission of the author of the code fragment and without fully acknowledging the author in the product documentation.

When the product is developed for this doctor's surgery, there would need to be a contract drawn up documenting who actually owns the developed product.

If it is written exactly to the doctor's specifications and they pay accordingly just for the time and skills to develop it, then they have ownership of the product not the programmer. In this case, it is not legal for the programmer to sell it, as it is not theirs to sell. The doctor's surgery owns the product, and may make changes to the software product whenever they feel it needs updating.

However, if the contract states that the programmer is developing a generic product for future sale, and the doctor's surgery will be involved in the initial design as a test site, and both parties clearly understand the conditions of this end-user agreement, then the surgery can not make changes to the product as it is legally owned by the developer who has allowed the surgery to pay for the use of their product. In this case, it would be accepted that the solution developed would be a generic solution appropriate for use by a large number of surgeries without necessarily meeting particular specific needs that are required by the original surgery. Also, proprietary information such as logos, fee structures etc that are owned by the original surgery could not be incorporated into the generic product being sold to other surgeries.



GROUP TASK Discussion

Do you agree that a combination of the prototyping and RAD approaches is the most suitable software development approach for the above project? Consider the suitability of both structured and agile approaches as part of your discussion.

SET 2A

1. A model of the general technique used to produce a software product is known as a:
   (A) software development cycle.
   (B) software development approach.
   (C) method of implementation.
   (D) prototype.

2. The software development approach where each stage must be complete before the next stage commences is called the:
   (A) prototyping approach.
   (B) end user approach.
   (C) structured approach.
   (D) RAD approach.

3. The end user approach often results in a compromise between:
   (A) developers and users.
   (B) requirements and functionality.
   (C) manual coding and wizards.
   (D) quality and reliability.

4. The acknowledgement that many requirements could not be known until design commenced has resulted in the:
   (A) prototyping and agile approach.
   (B) end user and structured approach.
   (C) prototyping and RAD approach.
   (D) RAD and structured approach.

5. Prototypes designed to promote discussion and refine requirements but not designed to be the final product are known as:
   (A) evolutionary prototypes.
   (B) structured prototypes.
   (C) functional prototypes.
   (D) concept prototypes.

6. Reducing development time is an aim of which development approach?
   (A) prototyping approach.
   (B) RAD approach.
   (C) structured approach.
   (D) end user approach.

7. Development teams comprising multi-skilled developers and users is a characteristic of the:
   (A) structured approach.
   (B) end user development approach.
   (C) prototyping approach.
   (D) agile approach.

8. The development of 4GLs was a major factor in the wide acceptance of the:
   (A) structured approach.
   (B) prototyping approach.
   (C) RAD approach.
   (D) agile approach.

9. The most significant and costly errors encountered during the software development cycle occur in the:
   (A) defining and understanding the problem stage.
   (B) planning and design stage.
   (C) implementation stage.
   (D) testing and evaluation stage.

10. It is vital to complete a software project as soon as possible even if this means some secondary requirements are not fulfilled. What is the most likely software development approach?
    (A) Structured.
    (B) RAD.
    (C) Prototyping.
    (D) End user.

11. List and describe the essential differences between the structured software development approach and the agile software development approach.

12. Rapid Application Development has become a recognised software development approach. Explain why RAD was not seriously recognised until the late 1990s.

13. A new accounting application is being planned to cater for the accounting needs of law businesses. It needs to run across various types of local area networks and must be able to interface with a large number of other software applications. The product will utilise a standard graphical user interface. Describe a software development approach or combination of approaches suitable for the development of this product. Explain the reasons why you chose that approach.

14. For centuries designers have been using prototypes. Explain how prototypes used for software development differ from those used in most other industries. Give examples to justify your answer.

15. 'Nobody writes algorithms anymore: now we've got software that writes applications for us'. Do you agree with this statement? Justify your answer and give examples.

CASE TOOLS IN LARGE SYSTEM DEVELOPMENT

Computer Aided Software Engineering (CASE) tools are used to assist and coordinate the activities involved in the process of software development. Any software tool that is used to assist developers can be classified as a CASE tool. Simple drawing programs can be used to create dataflow diagrams, system flowcharts or algorithms. Text processors could be used to create all forms of documentation. For larger systems, where a team of developers is at work, more complex CASE tools dedicated to software development are available. These tools provide a development environment that encourages a structured, organised approach to software creation.

Many integrated CASE tools provide all or some of the following functions:

- System modelling
- Data dictionary creation
- Production of documentation
- Automatic test data generation
- Software versioning
- Production of source code

Let us now examine some examples of CASE tools:

Oracle Designer

Oracle Designer is a CASE tool that specialises in system design, from creating a process model through various system modelling diagrams and finally to the creation of source code and user interfaces. This product is particularly suitable for the development of data-oriented applications. Oracle Designer uses an area, known as 'the repository' to store preferences used in the generation of the final source code and user interfaces. Code can be generated for many popular languages, including C++ and Visual Basic. The code created can then be loaded into the language's development environment for further modification before being compiled. Help files, for use with your application, can also be automatically generated from module definition information stored in the repository.

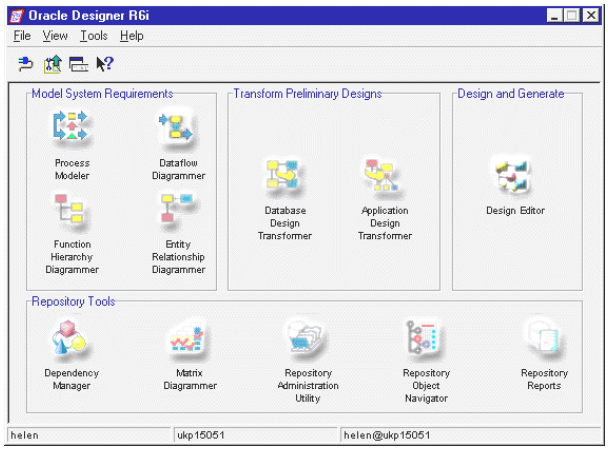


Fig 2.12
Oracle Designer's front panel.
The repository is used to set preferences and parameters used by the other modules.

This product is aimed at teams of developers, and hence it allows multiple users to access and modify aspects of the development of a product. Each object in the project contains its own versioning system. When a user modifies an object, Oracle Designer stores a record of the changes made and the user's name, as well as incrementing the version number of the object. Previous versions of objects are maintained, so unwanted changes can always be deleted.

AxiomSys

AxiomSys is a system requirements modelling CASE tool. AxiomSys can be used to generate dataflow diagrams for systems of all types, not just software systems. This tool allows for the creation of modularised dataflow diagrams together with data dictionary capabilities in regard to all dataflows used. Data items are defined using a simplified BNF (Backus-Naur Format). To enable monitoring of work still to be done on a particular project, AxiomSys provides graphical indications of work still to be completed.

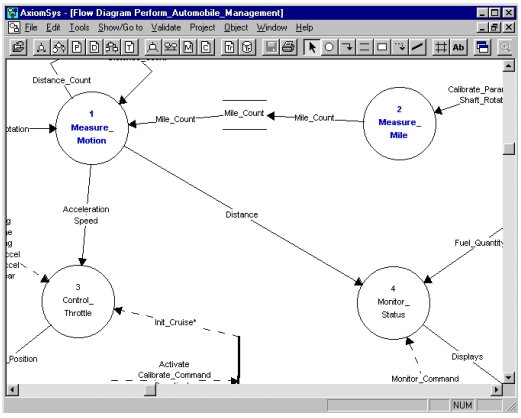


Fig 2.13
AxiomSys is used to create complex modularised dataflow diagrams.

DataFactory

DataFactory is a test data generation CASE tool. This tool is able to create random test data in large quantities. It is especially suited to testing large, data-oriented applications. Hundreds of thousands of records can be generated using data of user-specified data types. Products such as DataFactory allow real world testing of applications without the need for massive amounts of data to be manually input or scanned. Once the test data has been created you can have DataFactory alter dates so that leap years and other date-sensitive occurrences can be tested. DataFactory can write test data to existing databases. Any relationships and validation rules set up in the database will be observed during test data generation. Test data can also be written direct to files in various user-defined formats.

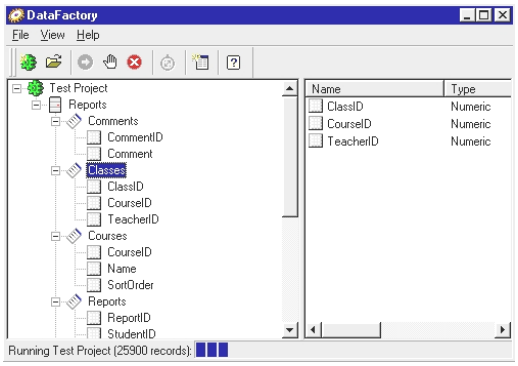


Fig 2.14
Using DataFactory to generate test data directly into an existing MS-Access database.



GROUP TASK Practical Activity
Search the Internet for CASE tools appropriate for the generation of dataflow diagrams. Find a shareware or demonstration version of one of these packages. Download it and create and print a simple dataflow diagram.



GROUP TASK Practical Activity
Find a test data generation program on the Internet. Install a demonstration version and use it to generate a file containing names and birth dates for as many people as the demonstration version will allow.



GROUP TASK Research
Today many integrated development environments (IDEs) include many of the functions of older dedicated CASE tools, for example, Visual Studio, Netbeans and Eclipse. Determine the CASE functions available in an IDE you have used.

METHODS OF INSTALLATION

Once a new software product has been produced it must be installed and then implemented on site. There are a number of methods of introducing a new system and each of these methods suits different circumstances. Usually installation will involve conversion from an old system that the new system is designed to replace, so often these methods are known as methods of conversion.

The four common methods of installation (or conversion) are:

- Direct cut-over
- Parallel
- Phased
- Pilot

We now consider each of these methods in turn.

DIRECT CUT-OVER

This method involves the old system being completely dropped and the new system being completely installed at the same time. The old system is no longer available. As a consequence, you must be absolutely sure that the new system is totally functional and operational. This conversion method is used when it is not feasible to continue operating two systems together. Any data to be used in the new system must be converted and imported from the old system. Users must be fully trained in the operation of the new system before the conversion takes place.

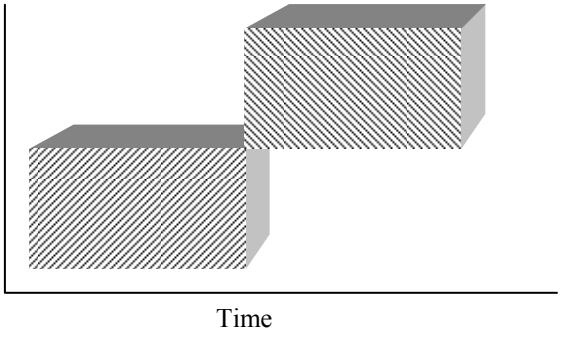


Fig 2.15
The Direct Cut-Over method of installation.



Consider the following:

A bank is introducing a new Automatic Teller Machine into many of its suburban branches. This new ATM includes a colour screen, which is also a touch screen. The software that controls the ATM has been thoroughly tested.

A direct cut-over method of installation is really the only method open to the bank. It would be impracticable to operate the old and new ATMs side by side. Even if this were possible, it would be ridiculous to expect customers to enter their details into both machines.



GROUP TASK Discussion
List and describe any potential problems you can envisage that may result from the sudden installation of these new ATMs. Think about issues from the point of view of both the bank and its customers.



Consider the following scenario:

A circus, which travels throughout Australia, is changing its ticket selling system from a manual system to a computerised system. Patrons purchase their tickets at a ticket booth just before they enter the big tent.



GROUP TASK Discussion
Is a direct cut-over conversion method appropriate for the above scenario? Explain your answer. Do all functions of the ticketing system need to be converted using the direct cut-over method? Discuss.

PARALLEL

The parallel method of installation (or conversion) involves operating both systems together for a period. This allows any major problems with the new system to be encountered and corrected without the loss of data. Parallel conversion also means users of the system have time to familiarise themselves fully with the operation of the new system. In essence, the old system remains operational as a backup for the new system. Once the new system is found to be meeting requirements then operation of the old system can cease. The parallel method involves double the workload for users as all functions must be performed on both the old and the new systems.

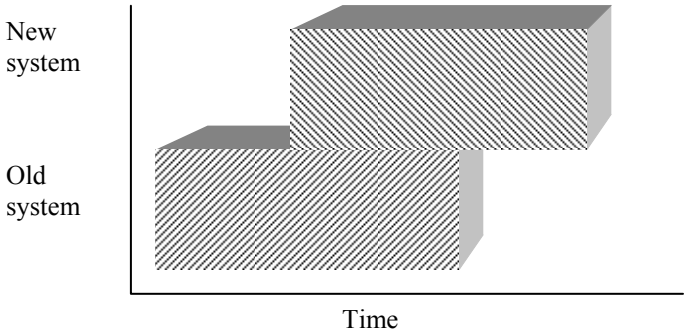


Fig 2.16
The Parallel method of installation.

This method of installation is especially useful when the product is of a crucial nature. That is, dire consequences would result if the new system were to fail. By continuing operation of the old system, the crucial nature of the data is protected.

From the software developer's point of view, parallel conversion can mean that both the old system and the new, upgraded system are operating at the same time but at different sites. Each individual site may have used a direct cut-over method of conversion. This situation is particularly common when the system is used by a large number of customers and they must be given time to commence operations using the new system. With a large customer base it will take some time to upgrade each site to the new product. A consequence of this situation is that the developer must continue to support previous versions of their product until the conversion is completed.



Consider the following:

The digital mobile phone network is now the only type of network available in large Australian cities. Digital mobile networks were introduced in Australia in the early 1990s, however the old analog mobile networks were only taken out of service in the late 1990s. Both systems operated together for some 5-10 years.



GROUP TASK Discussion
Why was it necessary for both analog and digital mobile phone services to be maintained for so many years?



GROUP TASK Discussion
Many users were happy with the old analog networks. Why do you think the mobile phone service providers decided to switch to digital networks?



Consider the following:

Five new computer-controlled life support systems have been purchased by a hospital. These life support systems are able to monitor a patient's temperature, blood pressure and various other vital signs. When an irregularity is detected, the medical staff are alerted electronically. Medical staff continue to manually monitor each patient's vital signs and record them on a paper chart on the end of each patient's bed.



GROUP TASK Discussion
Is it necessary for a patient's vital signs to continue to be manually monitored? Explain your answer.



GROUP TASK Discussion
The hospital decides it is no longer necessary for the manual system to remain in place. How would you feel about this decision if you were one of the patients, or a close relative or friend of one of the patients?

PHASED

The phased method of installation (or conversion) from an old system to a new system involves a gradual introduction of the new system whilst the old system is progressively discarded. This can be achieved by introducing new parts of the new product one at a time while the older parts being replaced are removed.

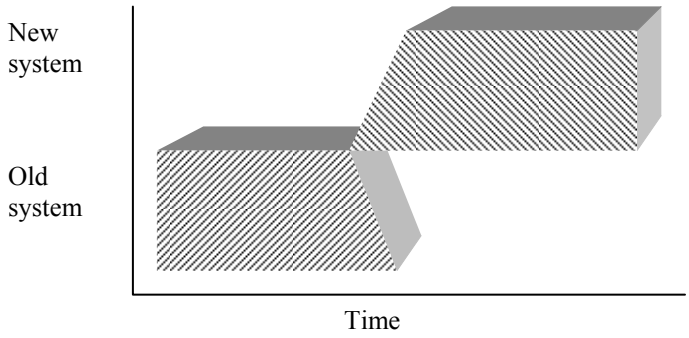


Fig 2.17
The Phased method of installation.

Often phased conversion is used because the product, as a whole, is still under development. Completed modules are released to customers as they become available. Phased conversion can also mean, for large businesses, that the conversion process is more manageable. Parts of the total system are introduced systematically across the business, each part replacing a component of the old system. Over time the complete system will be converted.



Consider the following scenario:

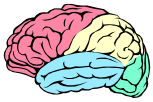
A software development business, called Chemsoft, specialises in software to support the operations of pharmacies. They currently have around 4000 chemists using their product. Chemsoft constantly works on upgrading components of their product. As each component is completed, it is distributed to chemists for installation. Component upgrades are sent around 3 times per year. A full upgrade of the product usually takes about 18 months.



GROUP TASK Discussion
It is not possible for Chemsoft to distribute and install each component at all sites at the same time. Discuss methods Chemsoft could use to implement the upgrade of each component. Assume nobody from Chemsoft has to be physically onsite to perform the installation.



GROUP TASK Discussion
Some chemists never quite get around to performing some of the upgrades. What sort of problems would this cause for the software developer Chemsoft? How could Chemsoft encourage their clients to implement the upgrades?



Consider the following scenario:

A large restaurant is introducing a new computer system. There are essentially four software modules that interface together to operate the functions of the restaurant: point of sale, accounting, wages and ordering/stocktaking.



GROUP TASK Discussion
If the restaurant decides to use a phased conversion method, how would this be achieved? Justify the order of installation for each of the four modules that make up the new package.



GROUP TASK Discussion
As part of the conversion the staff need to be trained to use the new system. The restaurant currently employs a total of 4 managers, 15 kitchen staff and some 20 table staff. Suggest how the training of all these staff members could be accomplished in an economical and efficient manner.

PILOT

With the Pilot method of installation (or conversion), the new system is installed for a small number of users. These users learn, use and evaluate the new system. Once the new system is deemed to be performing satisfactorily then the system is installed and used by all. This method is particularly useful for new products, as it ensures functionality is at a level that can perform in a real operational setting. The pilot method also allows a base of users to learn the new system. These users can then assist with the training of others once the system has been fully installed. The pilot conversion method can be viewed as the final testing of the product. Both the developer and the customer are able to evaluate the product in an operational environment prior to its full installation.

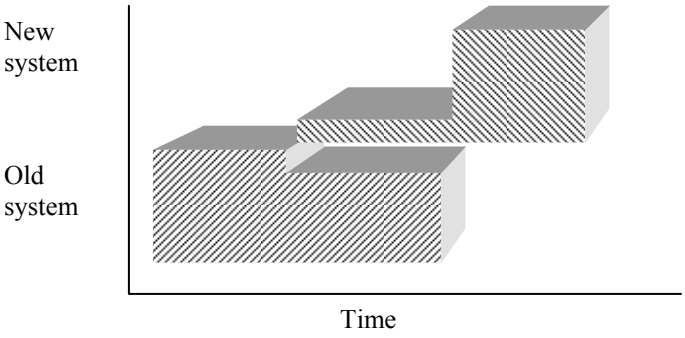


Fig 2.18
The Pilot method of installation.



Consider the following scenario:

A large company, with some 300 employees, is intending to change its Email software. They decide on a new Email software package. The package is installed and used by 15 employees for 3 months. Following this trial (or pilot) period, the new Email system is implemented on all employees' machines.



GROUP TASK Discussion
Why do you think the company decided to trial the Email package with 15 employees? Explain your answer.



GROUP TASK Compile a list of criteria
The 15 employees used for the trial were carefully chosen. Make up a list of criteria the company management may have used to select these 15 employees for the trial.



Consider the following scenario:

A large high school at present has handwritten reports for parents. They wish to computerise their reporting to parents to both improve the efficiency of their reporting and also to enhance the look of the reports. They decide on a commercial school reports package that seems to fulfil their requirements.



GROUP TASK Discussion
How could the school best pilot this new package? What sort of criteria could they use to evaluate the success of the pilot?



GROUP TASK Discussion
Many teachers at the school have only basic computer skills. How could the pilot method of conversion be used to increase the computer skills of the teachers at this school?



HSC style question:

(a) Define the term *parallel method of installation*.

(b) Describe characteristics of systems that make them suited to a parallel method of installation. Refer to example systems to illustrate your answer.

Suggested Solution

(a) The new system is installed and commences operation whilst the existing system continues to operate. That is, the new system and the existing system operate together for a period of time.

(b) Characteristics of systems suitable for parallel implementation include:

- The processing is critical to the organisation and hence it is desirable to ensure the new system operates correctly before removing the old system. For example, introducing a new patient monitor into an intensive care unit (ICU). Patients can be manually monitored for a period of time.
- It is reasonable to operate both old and new systems simultaneously. In the ICU example this is possible, however for a system that automates part of a production line this would be unworkable.
- Where the extra workload for users is reasonable. For instance a manual filing system can remain whilst an automated data entry system is introduced. If an existing data entry system was present then it would be unreasonable to enter the data into both the old and new systems, unless a utility was in operation to convert the data input into the format expected by the other system.



GROUP TASK Questions

Answer the above questions again for each of the remaining methods of installation, namely:

- Direct cut over method of installation
- Phased method of installation
- Pilot method of installation

SET 2B

1. Gradually introducing a new system onto a site is a feature of which method of conversion?
   (A) Direct cut-over
   (B) Parallel
   (C) Phased
   (D) Pilot

2. A new software product is implemented at a small number of sites. Once it is operating well, the remaining sites are converted to the new system. The method of implementation used is the:
   (A) direct cut-over
   (B) parallel
   (C) phased
   (D) pilot

3. The best reason for using a pilot method of conversion is:
   (A) to ensure all users are proficient before full implementation occurs.
   (B) as a final test of the product under actual operational conditions.
   (C) so that management can monitor the implementation more closely.
   (D) to protect the old system from possible corruption due to incompatibilities with the new system.

4. Direct cut-over would only be considered when:
   (A) thorough testing of the new system has been undertaken.
   (B) training is given to all users during or prior to the conversion.
   (C) it is not feasible for two systems to operate.
   (D) All of the above.

5. A phased method of implementation would be considered when:
   (A) the hardware necessary to implement the new system is unable to support the old system.
   (B) components in the new system share data dynamically.
   (C) no old system exists.
   (D) components of the new system can interact with data from the old system.

6. A parallel method of conversion should be used when:
   (A) testing has determined that the new system meets or exceeds all requirements.
   (B) the data is of a critical nature and system failure would have dire consequences.
   (C) it is vital to implement the new system as quickly as possible.
   (D) it is impracticable to operate both old and new systems simultaneously.

7. Users must be fully trained to use the new system when a ____ method of implementation is used?
   (A) direct cut-over
   (B) parallel
   (C) phased
   (D) pilot

8. Both old and new systems continue to be used for a period when a ____ method of implementation is used?
   (A) direct cut-over
   (B) parallel
   (C) phased
   (D) pilot

9. A poultry farm installs a computerised feeding and watering system throughout all its sheds. The owners have viewed the same system successfully operating on a number of other poultry farms. Which method of implementation would most likely be used?
   (A) Direct cut-over
   (B) Parallel
   (C) Phased
   (D) Pilot

10. The energy authority in a large city is implementing a new computer system throughout their branches. They decide to implement parts of the system progressively over a two-year period. Which method of implementation is being used?
    (A) direct cut-over
    (B) parallel
    (C) phased
    (D) pilot

11. Identify examples of CASE tools that would be used during each of the following stages of the software development cycle.
    (a) Defining and understanding the problem
    (b) Planning and designing
    (c) Implementation
    (d) Testing and evaluation
    (e) Maintenance

12. Make up a list of criteria that could be used to assist in the selection of users to participate in an initial pilot of a new software product.

13. Describe the advantages and disadvantages of the direct cut-over method of implementation compared to the other three methods.

14. The existing printer driver for a popular model printer is found to have a number of significant faults. An upgrade of the driver is written and is ready for use. Devise a strategy the printer manufacturer could employ to distribute and install this upgraded driver.

15. Investigations following a major air accident result in the rewriting of a software-based backup system that controls the flow of fuel to a jet's engines if the main system fails. The problem is unlikely to cause future accidents, however conversion is made mandatory for all airlines. The regulatory body controlling air safety needs to develop an installation strategy that ensures all aircraft are converted. Suggest, explain and justify a possible strategy.

CURRENT TRENDS IN SOFTWARE DEVELOPMENT

The information technology environment is becoming increasingly more complex. Multiple hardware platforms including a variety of mobile devices, diverse software development environments, countless desktop packages, complex multi-vendor systems, internet and web-based based software… and the list grows larger and larger all the time. It is becoming increasingly difficult for information technology departments to keep abreast of all these changes. It is even more difficult for them to be experts on such a diverse range of software products and the hardware on which it will execute.

Some of the more significant trends that have emerged include employment trends such as outsourcing and contract-based employment, and the changing nature of software applications to include web and mobile technologies such as cloud computing and apps for mobile phones. These applications are written in a diverse and large variety of programming language. We will consider each of these in turn.

EMPLOYMENT TRENDS

Employment prospects in the Information Technology Industry, and particularly in the area of software development, remain encouraging. There was a large annual increase in employment opportunities both globally and nationally during the five-year period from 1995 to 2000. Since 2000 employment in Information Technology has continued to grow albeit at a slower rate.

In the past employment as a software developer was largely based on experience. Today tertiary-qualified graduates are generally required to meet the demand for software development jobs. At present, there remain many people working in the Information Technology industry who hold qualifications in other areas and who have been paid by their employer to train as software professionals.

Outsourcing

As the software development industry has matured, businesses specialising in particular areas of software development have emerged. It is now unusual for businesses to write their own applications in-house: rather, they contract the services of a specialist software developer. This is known as 'outsourcing'. Outsourcing is not only applied to actually writing the software, but can be used for almost any task a business undertakes. For example, software support is often outsourced to a company specialising in that area. Outsourcing, although particularly common in the information technology industry, is common throughout all industries. For example, a lighting importer may outsource all their storage and distribution functions to a business specialising in that area. It is often cost effective to outsource to a specialist than to attempt the task in-house.

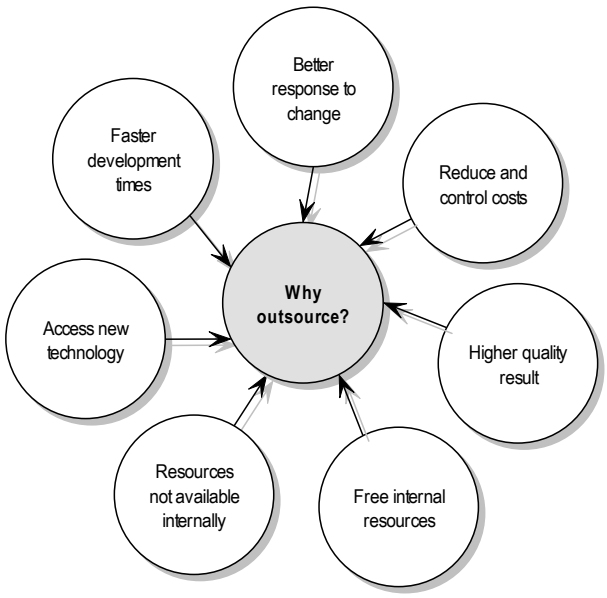


Fig 2.19
Some reasons companies choose to outsource.

Software development companies will also outsource work to specialists if they do not have the expertise or resources internally. If you examine most large software packages they will often acknowledge the work of other software developers as a part of the product's development. Often this information is contained on the 'Help' menu under 'About'.

Some reasons why businesses may choose to outsource software development include: to reduce and control costs; higher quality results from specialists; to free internal resources for other purposes; resources not available internally; access to new technologies; faster development times; and specialists respond better to change.

Outsourcing is used by both small and large businesses. Many businesses do not maintain an information technology department at all: they outsource their entire technology operation.



Consider the following scenario:

A courier company has for many years used two-way radios to communicate with its drivers. It is intending to computerise its operations, and each driver will have a small, dedicated computer mounted on the dashboard of their vehicle. The courier company will communicate work directly to each driver's computer.



GROUP TASK Discussion
Discuss the merits of the courier company outsourcing the development, training and ongoing support for this new system. Are there some aspects of this system that the courier company should operate internally? Explain.



GROUP TASK Discussion
Let's assume the above system has now been in use for a number of years. A few of the courier drivers are complaining that they always get the lower-paid jobs. As the software, via an outsourced specialist, performs allocation of jobs, how should the company respond to the drivers' complaints? If the job allocation is not fair and equitable, whose fault is this? Explain your responses.



Consider the following scenario:

A large financial institution currently has an internal software development team. The financial institution has just carried out a thorough investigation of all their technology resources using a prominent consultancy firm. The consultant's report strongly recommends to the financial institution's board of directors that they outsource all their software development and offer redundancy packages to members of the current internal software development team.



GROUP TASK Discussion
Why do you think the consultant's report may have recommended outsourcing the software development functions?



GROUP TASK Discussion
As it turns out, the outsourcing software development company subsequently employs many of the original team. Does this negate some of the advantages of outsourcing for the financial institution? Explain. What about the employees: how would you view this situation if you were one of these people?

Contract programmers

The nature of employment in the software development area is changing. Many positions of employment are short-term contracts to write particular software products. It is common for analysts and programmers to change employer often as they seek new contracts. Contracts can be with companies that are writing software for their own, internal use or with software development companies who are writing software for external customers.



Fig 2.20
Often software developers are on short-term contracts.

Websites have been setup to connect programmers with customers requiring software development work. Some of these websites utilise auctions whereby contract programmers compete to obtain project work. From the customer's perspective it is difficult to determine the expertise of individual programmers bidding for their project, particularly when such a diverse range of programmers from all over the world routinely bid for work. As a consequence many sites have developed rating systems based on the quality and timeliness of prior projects.



Consider the following scenario:

People involved in the development of software can be employed on short-term contracts. The employment contracts generally have a term of a few weeks but can last as long as twelve months.



GROUP TASK Discussion
What effect do you think this contract-based employment would have on the personal lives and finances of these employees?



GROUP TASK Discussion
List and describe the advantages of contract-based employment for both the employer and the employees.



GROUP TASK Research
Research a website which offers programming contracts. Determine the basis for awarding contracts and how the expertise of programmers can be determined by customers.



Consider the following:

The information technology industry is still relatively young when compared to other established industries. As a consequence, a problem exists in that the number of trained professionals with many years of experience is limited. Many training courses can quickly become outdated due to the rapidly changing nature of the industry, further compounding the problem.



GROUP TASK Discussion
With particular reference to software developers, discuss strategies software development professionals can employ to ensure their expertise remains current over a career spanning some 40-50 years.



GROUP TASK Discussion
Educational institutions are traditionally slow to adapt their courses to suit current industry practices. Suggest some strategies educational authorities could (or do) employ to ensure their courses remain current.

TRENDS IN SOFTWARE DEVELOPMENT

Environment in which developers work

Networks of computers, and in particular the Internet, have created a new and growing environment for software development. Developers from across the globe can collaborate on software projects. Many popular open source projects have contributors from a wide range of countries. Often these developers never meet in person and may not even speak the same language. Managing and combining the contributions from a wide variety of contributors would be impossible without suitable CASE tools. Many open source projects use Concurrent Versions System (CVS) which allows many different versions of each file to co-exist and many different developers are able to work on files concurrently.

Within software development companies teams of developers work closely together. They continuously collaborate to produce quality code. In the past each programmer was given a specific set of programming tasks to complete (often a list of subroutines together with apparently complete specifications). The programmers then worked on their assigned tasks in isolation. Today this is rarely the case, now developers are more than just coders. They are multi-skilled and work together with other developers and users as the development process progresses.



GROUP TASK Discussion
Often multiple programmers in different countries work on the same software project. Brainstorm a list of management issues that need to be addressed to ensure code is not lost or overwritten and to ensure the best code is included in the software application.



GROUP TASK Research
Using the internet, or otherwise, research a particular company to determine the typical tasks that software developers perform during a typical week of work.

Changing nature of applications

Recently the nature of software applications has undergone rapid change. In the recent past most software developers where writing applications for desktop computers or for servers. The recent changes are largely due to the widespread public acceptance and use of the internet and mobile devices. Software developers need to continually update their skills to take account of public demand for internet connected (or at least distributed) software applications.

Some examples of recent trends with regard to the nature of software applications are outlined below. Note that many of the terms are new and hence their meaning is still evolving. In addition many applications fall into a number of the following categories. For example, Facebook is an example of a web-based collaborative environment and is also a web 2.0 tool.

- **Web-based software**

The World Wide Web (WWW) is the significant part of the internet where web browsers access files stored on web servers. In the past most of the files were static HTML files and the web browser simply formatted and displayed these pages based on the HTML tags they contained. Today most web servers are able to generate dynamic content specific to each individual request from a user's web browser. The software running on the web server side generates the dynamic content and sends it to the browser using languages such as PHP and Perl. At the browser or client side, code can also be embedded within the HTML.

Languages such as JavaScript and ActionScript (within Flash files) are current commonly used client side languages. In 2011 a new version of HTML (HTML 5.0) was released which includes much of the functionality that previously required a third party embedded language. HTML 5.0 support will be included within most web browsers and hence it is expected to soon dominate in terms of active browser content.

A combination of both server and client (browser) code is needed to execute most web-based applications. Most Web 2.0 tools, cloud computing and even learning management systems are examples of web-based applications.

- **Learning objects**

A learning object is a collection of resources used to teach a specific chunk of knowledge. In real terms learning objects are an actual collection of resources, for example a science teacher might have a milk crate containing all the equipment needed for an experiment, together with worksheets, teacher notes and assessment items. Similarly, digital learning objects are collections of digital resources used to teach some chunk of knowledge.

Learning Management Systems (LMS) and Learning Content Management Systems (LCMS) are management systems for digital learning objects. An LCMS is used to produce and publish digital content which will form part of one or more learning objects. The LMS is used to deliver the learning objects to students. Delivery is commonly through a web browser and can be over the educational institution's LAN (intranet) or it can be delivered over the internet. In addition an LMS can monitor student progress, deliver assessment items and tests as well as record student results within its linked database.



GROUP TASK Research
Research to determine the LMS and/or LCMS used by your school or by other schools. Outline how this application is used within the school.

- **Widgets**

The term widget has a range of meanings. In its purest form a widget is a graphical user interface (GUI) element, for example, text boxes, menus, radio buttons and checkboxes are all examples of GUI widgets. The term widget also refers to desktop widgets. These are small applications that commonly reside on the desktop of personal computers. Common examples include clocks, weather forecasts, image slideshows and small games. Often these desktop widgets utilise the internet to update their content at regular intervals. Web widgets are another type of widget that is often included as part of a web page. Many social networking sites enable users to include small web widgets on user's pages. Content from other sites (such as a twitter feed or news feed) and small interactive games are common examples.

- **Apps**

The term "App" was once simply a shortened form of the word application. Since the advent of advanced mobile phones such as Apple's iPhone and Android phones the term App has evolved to refer to applications executed on mobile phones. Due to restricted RAM and secondary storage, Apps are generally much smaller applications compared to their desktop computer alternatives. Currently, on the iPhone all Apps must be downloaded from Apple's App Store. This restriction means all developers of iPhone Apps must register with Apple and can only distribute their Apps through Apple's App Store. Currently Apple receives 30% of all App Store revenue. Such restrictions do not exist for Android phones, however Google's Android Market is the most popular site for downloading Android phone Apps as Google's "Market" application is preinstalled on virtually all current Android phones. Google owns the Android operating system, however in 2008 it was released under an open source licence. Nevertheless Android Apps can be obtained from a variety of sources, including direct from the developer's website.

- **Applets**

An applet is simply a small application. For example, most operating systems include a variety of applets, such as notepad and paint, which perform basic tasks but are not as fully featured. For instance notepad includes basic text processing but is lacking much of the functionality found in a full-featured word processor. Many applets run within a web page or within another application. Plug-ins for web browsers such as QuickTime and Adobe's Flash Player are examples of applets. Many websites also include Java Applets which allows the browser to run Java bytecode using it's JVM (Java Virtual Machine). Java bytecode is particularly popular as it can be executed on a range of different operating systems.

- **Web 2.0 tools**

The term Web 2.0 was first used back in 1999, but gained popularity in 2003 largely due to Tim O'Reilly hosting the first Web 2.0 conference. Web 2.0 tools are all about user participation and collaboration. They enable users to discuss, share and contribute in a more human and intuitive manner than was possible with previous web tools. In essence Web 2.0 promotes users creating data and then sharing and combining their data across a variety of applications. For instance, including Flickr

photos within a Facebook page or linking photos to locations on Google maps. In terms of technology there is no real difference between earlier tools and those known as Web 2.0 tools. From a software development perspective Web 2.0 is about the nature of these tools and how they are used rather than any technological difference.

- **Cloud computing**

Cloud computing refers to software as a service (SaaS). The application runs over the internet, some parts executing on the client with most of the application running on the remote server. Commonly the application operates within the user's web browser but includes many features found in the equivalent desktop application. The term "cloud" refers to the unknown nature of the devices providing the service. Often the internet is represented simply as a cloud shape indicating data and information flow into and out of the cloud with no need to understand what occurs within the cloud. Cloud applications are similar as the user doesn't need to know where execution occurs or where data is located to utilise the service. Although many cloud applications allow documents to be saved locally, most often documents are stored on remote servers. This allows traditional documents such as word processor and spreadsheet files to be accessed and edited by multiple users.

- **Mobile phone technologies**

During the period 2000 to 2010 mobile phones have progressively included more and more features. Today most mobile phones include digital cameras, internet connectivity using both local 802.11 access points and 3G networks, Bluetooth and also GPS receivers. All these connectivity and other hardware features have resulted in an ever increasing number of innovative Apps coming onto the market. It is now common for social networking and other web-based applications to be routinely accessed using mobile phones. GPS functionality is being included in many Apps so that the user's current location can be used to filter received data and mark the location of photos and videos. The name "mobile phone" is now somewhat redundant as many users rarely use their phone to actually make phone calls, preferring to use text messages and various internet communication Apps to communicate. The number of mobile phone users worldwide continues to grow rapidly. Therefore, for software developers, writing Apps for mobile phones that utilise the various connectivity features in innovative ways has enormous potential.



GROUP TASK Discussion
List Apps installed on mobile phones owned by your classmates. Identify any communication features used by each App.

- **Collaborative environments**

A variety of software applications are emerging which aim to allow distributed workers or friends to communicate as if they were working physically alongside each other. Social networking sites and cloud computing allow collaboration in a limited manner. For instance, real time chat and the ability for users to edit a document simultaneously. However real collaboration requires voice and real time video interaction. There are now Voice over IP (VoIP) systems that provide high quality audio between teams of workers. Applications such as Skype add video, however video quality remains an issue. Many large software companies are working on high quality systems which will improve the potential uses of collaborative environments. Some systems are based on virtual world systems, where the users create their own avatar character who is able to move about the virtual world.

SET 2C

1. One of the most common reasons for outsourcing software development work is:
   (A) to increase a company's market share.
   (B) because the expertise and resources are not available in-house.
   (C) that the nature of employment for software developers has changed.
   (D) to save the company purchasing expensive development software.

2. Which of the following is true of COTS package-based products?
   (A) Performance is generally lower than traditionally developed products.
   (B) Performance is generally higher than traditionally developed products.
   (C) Cost of development is higher than traditionally developed products.
   (D) Development is restricted to large business applications.

3. Important employment trends for those working in software development include:
   (A) Increasing use of software on mobile devices.
   (B) Contract-based employment.
   (C) The ongoing global expansion and use of the internet.
   (D) All of the above.

4. Which of the following applications is used to create digital learning objects?
   (A) LMS
   (B) CMS
   (C) LCMS
   (D) Milk crates

5. A computer teacher develops a computerised mark book using a spreadsheet program. The mark book can be accessed from any computer linked to the school's LAN. This is an example of:
   (A) using a COTS package.
   (B) outsourcing.
   (C) network software.
   (D) a very smart teacher.

6. Cloud computing is often known as:
   (A) web-based computing
   (B) social networking
   (C) software as a service
   (D) Applets

7. Elements used on graphical user interfaces are often known as:
   (A) Apps
   (B) Learning objects
   (C) Applets
   (D) Widgets

8. A software company contracts an artificial intelligence software specialist to develop an optical character recognition module for their new scanning product. This is an example of:
   (A) contract-based employment.
   (B) outsourcing.
   (C) using a 4GL.
   (D) on the job training.

9. The continuing evolution of new computing devices has ramifications for employees in the software development industry. Which of the following is true?
   (A) Software development personnel need to be experts on network software to maintain their positions.
   (B) Training needs to be ongoing throughout the careers of those employed in the industry.
   (C) Computer systems need to be built that can perform all the tasks of the developer.
   (D) Soon there will be no place for small software development companies as large, multi-national companies will take over.

10. Which type of application allows and encourages users to create, share and combine data over the internet?
    (A) web-based software
    (B) Apps
    (C) Learning objects
    (D) Web 2.0 tools

11. Provide specific examples of applications that would be classified under one or more of the following headings.
    (a) Web-based software
    (b) Software for managing delivery of learning objects
    (c) Desktop widgets
    (d) Apps
    (e) Web 2.0 tools
    (f) Cloud computing

12. A software development business that specialises in artificial intelligence has just won a contract to produce the grammar checker for a major new word processor. The terms of the contract precisely specify the requirements for this grammar module. Because this is such a large contract and the time given for completion of the project is restricted, the business decides to employ 5 new programmers on short-term contracts. Why do you think the development of this grammar module was given to another business rather than being developed in-house? Do you think it is reasonable to employ programmers for a short period to complete contracts such as this? Explain your answers.

13. Humpty Homewares is a major retailer of furniture and electrical appliances. They currently have some 30 stores around the country. Humpty Homewares wish to commence marketing their products on the World Wide Web. An initial feasibility study indicates that the majority of customers, although happy to browse for furniture and appliances on the web, like to come into a local store to make the final purchase. Currently the head office has an information technology department supporting their existing computer systems. Outline the software development skills required to develop this new system. Should Humpty Homewares develop this system in-house?

14. A small husband-and-wife business markets and sells essential oils. They sell their oils at weekend markets and country shows as well as by mail order to repeat customers. The business has progressed to a level where they feel they need to automate their ordering, invoicing and mail order functions. The husband and wife have no experience with computers. You, as a software developer, have been asked to advise the couple on how to proceed. What advice would you give them? Assuming you are going to use COTS packages for this project, describe the nature of the packages you would use.

15. Use a search engine and the Internet to find statistics in regard to salaries paid to software developers. Examine both system analysts and programmers from Australia and overseas. Convert any overseas currencies to the equivalent in Australian dollars using a bank's website.



HSC style question:

Question 1.

Identify the essential tasks performed during each stage of the software development cycle when using a structured software development approach.

Question 2.

Large software development projects often fail to meet their initial time and budgetary constraints. In many cases these projects fail to deliver a final operational software product.

Outline likely reasons for such failures.

Question 3.

The RAD (rapid application development) and prototyping approaches to software development encourage user feedback and interaction throughout the development process.

Describe strategies and techniques that can be used to encourage user feedback and interaction when using the RAD and/or prototyping approaches to software development.

Suggested Solutions

Question 1.

Defining and understanding the problem

- Communicate with the client to define what needs to be done (input, processing, output) and the boundaries of the problem to be solved
- Consider relevant data structures and standard algorithms that might be relevant
- Initial systems modelling to better understand the current and proposed systems
- Use of a prototype to better understand the requirements of the client
- Define the criteria for product quality, and ensure processes are in place to ensure these criteria will be addressed appropriately

Planning and designing solution

- Creation and desk-checking of relevant algorithms for custom designed logic
- Identification of relevant off-the-shelf products that can be ethically modified
- Identification of relevant library routines suitable for incorporation
- Production of refined systems modelling and technical documentation
- Interface design in consultation with the client
- Selection of the programming language and approach

Implementation

- Generate the code required and remove syntax and logic errors
- Use debugging techniques to systematically remove logic errors
- Document the software solution (for both users and future maintenance)
- Acquire, install, configure and test required extra hardware

Testing and evaluating

- Performs module, program and systems testing to ensure that the product will run successfully once installed
- Include the use of live and volume data when systems testing
- Report on the testing process
- Evaluate the system to ensure that all requirements have been met, in discussion with the client

Maintenance

- Modify the code to meet changing requirements
- Document the changes, inside the program / module and in external documentation as appropriate

Question 2.

There are many reasons why a software project fails. These can include the following:

- Time money and resources not allocated to correctly determining the problem to be solved.
- Possible poor interview techniques meaning that clients' needs are not heard, understood or documented correctly
- Poor specifications for programmers
- Inadequate communication with the client on an on-going basis as the product develops
- Poor selection of technology used to develop / deliver the product
- Poor estimation of the time required to complete subtasks
- Poor allocation of resources to defined tasks (people are not skilled appropriately, or are moved around tasks unnecessarily, or not given access to required resources when they are required)
- Inadequate time allocated for testing, or inappropriate testing procedures used
- Poor data imported into the new system
- No defined criteria or processes for quality assurance

Question 3.

RAD makes use of predefined libraries and routines so that with minimal effort, a developer can quickly produce a working solution with interfaces that are usually easily editable using drag and drop facilities. This means that the developer can work with the client to make quick and easy modifications that reflect the client's feedback once they are shown a working version. This process will be ongoing as the product develops and efficiently produces a product that is acceptable to the client with the expected functionality.

The prototyping approach is usually more concerned with the development of interfaces that are relevant and appropriately reflect the needs of the client, and so requires the client and developer to work together in the initial stages of the product development to iteratively refine the initial prototype design. This ensures that the developer develops a good understanding of what the client is expecting from the product before the product starts being developed, as well as ensuring that the client has a good appreciation of what the final interfaces will look like giving them a good feel for how the product will appear to the user.

CHAPTER 2 REVIEW

1. The correct sequence of stages in the software development cycle is:
   (A) Defining and understanding the problem; planning and design; implementation; testing and evaluation; maintenance.
   (B) Defining and understanding the problem; implementation; planning and design; testing and evaluation; maintenance.
   (C) Defining and understanding the problem; planning and design; implementation; maintenance; testing and evaluation.
   (D) Planning and design; defining and understanding the problem; implementation; testing and evaluation; maintenance.
2. The structured approach to software development is also known as the:
   (A) prototyping approach.
   (B) RAD approach.
   (C) waterfall approach.
   (D) agile approach.
3. Often a full set of requirements cannot be created until the design stage has commenced. Which approach to software development acknowledges this situation by allowing for changing requirements?
   (A) Prototyping approach.
   (B) RAD approach.
   (C) Structured approach.
   (D) End user development.
4. Which iterative software development approach uses multi-skilled teams which include a representative user?
   (A) Structured approach
   (B) End user approach
   (C) Prototyping approach
   (D) Agile approach
5. Which software development approach that aims to get a product completed in the shortest space of time, even at the cost of not fulfilling some of the requirements?
   (A) prototyping approach
   (B) RAD approach
   (C) structured approach
   (D) end user approach
6. A prototype that is continually refined until it becomes the final product is known as a(n):
   (A) RAD prototype.
   (B) concept prototype.
   (C) experimental prototype.
   (D) evolutionary prototype.
7. The difference between a software development approach and a method of installation is:
   (A) A software development approach describes how the final product will be installed on the end user's site. A method of installation is the methodology used during the creation of a new product.
   (B) A software development approach is the methodology used during the creation of a new product. A method of installation describes how conversion to the final product will occur at the end user's site.
   (C) A software development approach is the methodology used during the creation of a new product. A method of installation is the way the designed solution is converted into a programming language for use on a computer.
   (D) A software development approach is determined by the programming language being used. A method of installation is independent of the programming language used.
8. A software product contains a number of self-contained modules. The modules are installed on sites in a particular sequence over a 12-month period. This is an example of:
   (A) parallel installation.
   (B) direct cut-over installation.
   (C) phased installation.
   (D) pilot installation.
9. Errors detected later in the software development cycle become increasingly expensive to correct. This is particularly the case when using the:
   (A) prototyping approach.
   (B) RAD approach.
   (C) structured approach.
   (D) agile approach.
10. CASE tools are:
    (A) packages that can be customised to perform more specific tasks.
    (B) used to create object code from system diagrams.
    (C) software tools that assist in the software development process.
    (D) used by large software developers to ensure that team members comply with set standards.

11. A software development company has decided to create a new CASE tool both for their own, internal use and to be distributed and sold to other developers. They are targeting companies that specialise in the development of mobile phone apps using an agile approach. List and describe some of the features that should be included in this package.

12. A new computer game is being considered for production by a large software development house. The game will be a 3D action-type adventure game based on a recently released movie. It will include the ability to play against opponents online. The game will be released for a number of different platforms, including popular game consoles, personal computers and also a free cut-down version for iPhones and Android phones.

    (a) Decide on a software development approach appropriate for the development of this product. Justify your choice.

    (b) Describe issues that are likely to occur when implementing the game on multiple platforms.

13. For each of the methods of installation presented in this chapter, list the main features and give an example of a scenario where each method of installation would be appropriate.

14. The nature of jobs in software development is changing: some jobs are full-time, others are part-time or casual, and many jobs are now contract-based. Examine newspaper classifieds or use a search engine on the World Wide Web to find jobs in software development. Make up a table showing company name, job profile, salary and nature of employment.



15. Many tertiary courses in the software development area are degrees in Computer Science, Information Systems or Software Engineering. Research a variety of tertiary courses on offer using either the University Admission Centre's (UAC) printed guide or their website. UAC's web address at the time of printing was http://uac.edu.au. Make a list of these courses together with a brief description of their content. Also include the ATAR required for entrance based on the previous year's intake of students.

In this chapter you will learn to:

- evaluate the extent to which a proposed system will meet user needs
- evaluate the effectiveness of using existing software
- identify the parts of the proposed system that require software to be designed and developed
- identify a relevant approach for a given problem
- develop and interpret design specifications from a user's perspective
- recognise the difference between the user's and developer's perspectives and the communication issues that may arise
- differentiate between forms of systems documentation and the purposes for which each is used
- describe a system by interpreting its diagrammatic representation
- create a diagrammatic representation for a system using appropriate modeling tools
- effectively communicate with users regarding a proposed software solution
- identify a range of criteria on which the quality of the product will be judged
- identify relevant processes for a given criterion that will result in a quality product

Which will make you more able to:

- differentiate between various methods used to construct software solutions
- identify and evaluate legal, social and ethical issues in a number of contexts
- construct software solutions that address legal, social and ethical issues
- identify needs to which software solutions are appropriate
- apply appropriate development methods to solve software problems
- apply a modular approach to implement well structured software solutions and evaluate their effectiveness
- apply project management techniques to maximise the productivity of the software development
- create and justify the need for the various types of documentation required for a software solution
- select and apply appropriate software to facilitate the design and development of software solutions
- assess the skills required in the software development cycle
- communicate the processes involved in a software solution to an inexperienced user
- use and describe a collaborative approach during the software development cycle
- develop and evaluate effective user interfaces, in consultation with appropriate people.

In this chapter you will learn about

Defining the problem

- identifying the problem
  - needs of the client
    - functionality requirements
    - compatibility issues
    - performance issues
  - boundaries of the problem

Issues relevant to a proposed solution

- determining if an existing solution can be used
  - social and ethical considerations
  - consideration of existing software products
  - customisation of existing software solutions
  - cost effectiveness
  - licensing considerations
- selecting an appropriate development approach if there is no appropriate existing solution

Design specifications

- specifications of the proposed system
- developer's perspective in consideration of:
  - data types
  - data structures
  - algorithms
- user's perspective
  - interface design
  - social and ethical issues
  - relevance to the user's environment and computer configuration

System documentation

- representing a system using systems modeling tools, including:
  - IPO diagrams
  - context diagrams
  - data flow diagrams (DFDs)
  - storyboards
  - structure charts
  - system flowcharts
  - data dictionaries
- algorithms used to document the logic in modules and subroutines
- test data and expected output

Communication issues between client and developer

- the need to consult with the client
- the need to incorporate the client's perspective
- the need for the developer to enable and consider feedback
- the need to involve and empower the client during the development process

Quality assurance

- the need to explicitly define the criteria on which the quality of the product will be judged
- putting in place management processes to ensure that quality criteria will be met
- an ongoing process throughout development to ensure the quality criteria will be met

3

DEFINING AND UNDERSTANDING THE PROBLEM

Defining and understanding the problem is the first stage of the software development cycle. In many ways, this initial stage is the most important. Formally identifying the precise problem to be solved and determining if an existing solution is viable or a new solution is needed are vital steps that are often overlooked and perceived as trivial by some large software developers. Studies have shown that some 30% of software projects are abandoned well into the development cycle. The majority of these cancelled projects are well over budget and clearly were not feasible from the outset. Software developers must take steps to carefully define and understand the problem before moving onto the planning and design phase.

In this chapter, we look at important considerations when identifying the precise problem to be solved. We examine how to identify and specify the problem so it meets the needs of the client – including the importance of client-developer communication. We also look at different methods of documenting and modelling the proposed new system. Finally we consider quality assurance. Quality assurance is an ongoing process during development which monitors and assesses the standard of the developing solution against a set of quality criteria.

DEFINING THE PROBLEM

Problems can come from a number of different sources. It may be that an outside company is looking to contract a software developer to construct a solution for their business. It could be a new idea for a product. In either case, the problem itself needs to be defined precisely so that both the developer and the client understand what will be done.

Some questions that need to be asked to help define and identify the problem are:

- What are the client's needs which will be met by this product?
- Compatibility issues with other existing software and hardware?
- Possible performance issues, particularly for internet and graphics (or video) intensive systems?
- What are the boundaries of the new system?

These questions need to be answered to clearly identify the problem prior to creating a set of design specifications.

NEEDS OF THE CLIENT

A need is an instance in which some necessity or want exists. The implication is that some form of solution is required to meet this need. Without actually articulating the needs clearly, it will be difficult to develop a clear picture of the precise problem to be solved. In most cases the needs of the client are the primary reason software projects

are developed. Often such needs are initially expressed as general statements, such as "We need to be able to monitor sales made at each of our stores". Clearly such general statements require significant consultation to determine how such monitoring can be performed. Through consultation and analysis a set of functionality requirements is created. Meeting these requirements is the central aim of the entire software development process.



Consider the following:

A window cleaning business, known as 'Mr Window Cleaner', has just completed an advertising campaign during which they have dropped 10,000 fridge magnets in letterboxes throughout the northern and eastern suburbs of Sydney. They claim, on the magnets, that a window cleaner can be on the job within 3 working days. The magnet drop is so successful that they are inundated with work. They are so busy that they are unable to even call back each potential customer within 3 days, let alone get a window cleaner to the job.

Let us consider the needs from this scenario.

The potential customer needs:

- To have their windows cleaned professionally
- To be contacted to organise a time for the work to be done
- To have the work completed within 3 working days
- To be charged an acceptable fee

The business 'Mr Window Cleaner' (the client) needs:

- To make a profit
- To fulfil customers' needs
- To contact customers quickly
- To complete each job within 3 working days
- To continue to service existing customers



GROUP TASK Discussion
A software developer may immediately see possible methods of meeting these needs. For example, setting up a computerised voice mail service to track client enquiries, developing a database of clients, automated scheduling of work, etc. The software developer is really expressing their 'need' to be part of the solution. Is the software developer ultimately going to meet the needs of 'Mr Window Cleaner' and their customers by implementing systems such as those mentioned above? What do you think? Consider other non-software alternatives open to the business as part of your discussion.



GROUP TASK Discussion
Do the customers really require their windows cleaned within 3 working days? Suggest ways of determining if meeting this need should be a high priority for the business.

Often needs are expressed in non-specific, even emotive terms. For example, we need to improve customer relations, or we want to improve our turnaround times. Analysing needs is an important aspect of beginning to define the problem and for large systems a professional systems analyst is used. A number of tools and techniques are available for accomplishing the analysis.



Systems Analyst
A professional who analyses systems, determines requirements and designs new information systems.

- Surveys - Completed by all key personnel. For example, managers, IT personnel, users and clients. Survey's are an excellent tool for gathering information from large numbers of people, however they often limit the detail and explanations participants are able to provide.
- Interviews - Personal interviews will allow participants to more freely express their needs. Detailed responses and explanations are possible, however performing interviews is labour intensive and hence expensive.
- Time management studies - How much time is really spent on each specific function. Actually watching and recording the time, nature and order of tasks as they are performed will often highlight priority areas.
- Business analysis - Examining different aspects of a business' activities in search of areas where improvement can be made. Often such analysis will include a cost benefit analysis to ensure a profitable return on the software investment is likely.

Many developers have great ideas for new products. This is all well and good, but is there a need for this product? Developing a new product of any type without establishing a need for the product is like putting the proverbial cart before the horse. Many great inventions have remained as ideas because no market or need exists for the product. Conversely, many fine products appear which fail economically because they don't meet the needs of the market place.



Consider the following:

A software development company is exploring the possibility of producing a computer controlled, in-home system. This system will use a personal computer to control the activation and operation of various electrical appliances within the home. It will be able to control the home's security system, lights, air conditioning, TV, video and various kitchen appliances. For example, when you leave the house the system will put all appliances into a sleep mode, and the air conditioning will automatically cool the house prior to your normal time of arrival in the evening.



GROUP TASK Discussion
How could the software development company establish if there was a need amongst potential customers for a system such as this?



GROUP TASK Debate
The company finds that potential customers don't really understand how the new computer controlled system can be of benefit. Should the company attempt to educate the public about this product and hence, hopefully, create a need, or should they cancel the project?

Functionality requirements

Functionality requirements describe what the system will do. They are what you are aiming to achieve. Requirements are statements that if achieved, will allow needs to be met. The requirements of a system give direction to the project. Throughout the design and development of a system the functionality requirements should continually be examined to ensure their fulfilment. The final evaluation of a project's success or failure is based on how well the original requirements have been achieved.



Requirements
Features, properties or behaviours a system must have to achieve its purpose. Each requirement must be verifiable.

To be able to evaluate the degree of success or failure of requirements requires that the requirements be stated in such a way that they can be readily verified. In terms of software development, this often means they must be quite scientific or mathematical statements. For example, a requirement may be: 'The system will be able to redraw the screen at least 12 times per second' or '95% of orders will be filled within 24 hours'. Each of these requirements can be precisely tested.



Consider the following:

A small mail order business has grown to a stage where it is no longer viable to use a manual ordering system. The business has established a list of needs as follows:

1. Orders must be processed more efficiently.
2. Overdue accounts need to be dealt with more effectively.
3. Items on backorder should always be filled before new orders.
4. Customers need to feel confident their personal details will remain confidential.
5. Personal attention to customers must be maintained.

After further discussion and analysis a set of functionality requirements is developed as follows:

- 90% of orders will leave the premises within 48 hours. Customers whose orders are not filled in this time are contacted by phone.
- Overdue accounts are generated automatically after 30 days.
- Accounts over 60 days are contacted personally. If a resolution is not found they are referred to a collection agency.
- Stock to be entered into the computer system before it is placed in the warehouse.
- Backorders are always checked before items are allocated to orders.
- Password protection on the customer details file.



GROUP TASK Discussion
Describe the essential differences between the above needs and the functionality requirements that follow.



GROUP TASK Discussion
Can you suggest further requirements that may assist in ensuring personal customer attention is maintained?

Compatibility Issues

Software of various types runs on a variety of operating systems, browsers, hardware configurations and even a range of different devices, such as smart phones and tablet computers. Users are able to configure their device in a variety of manners, for example screen resolution, colour depth and font sizes. Today many software products require and use internet or LAN connections. Such connections operate at varying speeds and will often encounter errors or even complete loss of connectivity. When designing software developers must ensure their products are compatible with a wide range of likely user devices and network conditions.

Some examples of common compatibility issues include:

- Problems with different versions of the intended operating system, particularly as users upgrade their operating system.
- COTS products which experience issues when the underlying application is updated.
- Not all graphics cards are supported by many graphics code libraries.
- Screens which do not resize to support different screen resolutions.
- Labels for screen elements which overlap or are no longer aligned correctly when fonts are enlarged.
- Loss of server connection causes fatal errors over wireless networks.
- Browsers which do not correctly implement HTML standards set by the W3C (World Wide Web Consortium).

Any existing hardware, software and communications systems must be documented and then taken into account when defining the problem to ensure the solution will be compatible with these existing resources.

Performance Issues

Often the specifications of the computers used by developers far exceed those likely to be present in a typical user's computer. In addition multi-user applications and applications that access large files and databases will perform very differently under real world conditions. Testing environments and actual tests should simulate real world conditions. We consider system testing in detail as part of chapter 6, however all testing, including performance testing should be ongoing throughout development.

When defining and understanding the problem it is common for performance requirements to be included. For example, "Online payments must be approved within 5 seconds" or "The application will load in less than 3 seconds". For custom systems written for known hardware and communication lines it is possible to ensure such time sensitive requirements are met with reasonable certainty, however when developing for a broad and largely unknown audience it is much more difficult. Nevertheless it is critical that developers use the most efficient algorithms for the task. If processing has the potential to take some time then visual aids should be used on the user interface to indicate the likely time required. In some cases time consuming processing can be performed in the background whilst the user performs other tasks.

Some common examples of performance issues include:

- The computer appears to be not responding after some function has been initiated. In fact it is busy processing a time consuming task.
- Users experience poor response times. This is often seen with networked applications where during data entry users spend most of their time waiting for the application.

BOUNDARIES OF THE PROBLEM

Boundaries define the limits of the problem or system to be developed. Anything outside the system is said to be a part of the environment. The system interacts with its environment via an interface. Input and output to and from the system takes place via an interface. The keyboard provides an interface that allows humans to input data into a computer system. An internet service provider (ISP) provides an interface between computers and the internet.

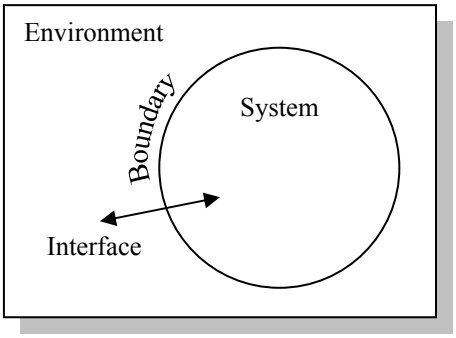


Fig 3.1
The boundary defines the limits of the system. Interfaces allow communication with the environment.

It is vital to determine the boundaries of a problem to be solved. Determining the boundaries is, in effect, determining what is and what is not a part of the system. Items that are not part of the system, that is items that are in the environment, need to be considered if they have an influence on the system. For items in the environment to influence the system there must be an interface with the system. Items in the environment can affect the system, however the system cannot alter its environment.

When defining a problem it is important to define the boundaries for the problem so that the customer has realistic expectations of the limits of the new system.



Consider the following:

A company that specialises in the production of custom kitchen cupboards is computerising its cabinet making processes. Each cabinet-maker will have a dedicated terminal that will provide him or her with details of the next cupboard they are to construct. Once construction is completed, the tradesperson enters the time of completion into his or her terminal and awaits allocation of their next job. The terminals use a proprietary operating system. You, as a software developer, are employed to develop the software for this system.

There are a number of limitations or boundaries apparent to you as a software developer, namely:

- Control over the dimensions of cupboards delivered via your software to the cabinet-makers. The software can only send the dimensions entered by the kitchen design staff.
- Ability to determine the time taken to build individual cabinets is dependant on the cabinet-makers efficiently entering jobs as complete at the correct time.
- As the terminals to be used by the cabinet-makers use a proprietary operating system, the functioning of the software is dependant on the capabilities of this operating system.



GROUP TASK Discussion
Explain how each of these boundaries may affect the overall success of the final system. Use examples of possible scenarios to illustrate your answers.

ISSUES RELEVANT TO A PROPOSED SOLUTION

Although a new solution may well meet the current identified needs there are other areas that should be considered prior to investing time and money in the development of a new software product. Will a new solution provide enough advantages to the organisation to be worth the time, effort and money involved in its development and implementation?

Points to consider from the client's perspective include:

- Will the existing system be able to perform the required tasks in the foreseeable future? If so, then maybe the new system should be cancelled until a later date.
- Will the proposed new system meet future needs? If not, then maybe the requirements need to be changed to reflect longer term needs.
- Have existing similar solutions been examined? Perhaps a suitable solution already exists or an existing software package can be customised.
- How crucial is the new system to the total organisation? Perhaps resources would be better utilised in other areas.

Some points to be considered by the software developer include:

- What expertise is required to complete this project? Do we already have this expertise and if not then can we get it?
- What resources are required to develop this project? Do we have these resources and if not can we get them?
- Will we be able to support this product in the future?
- Can we retrain/redistribute existing staff and resources to this project? How would this affect other projects?

Different methods and approaches to the solution of a problem should be considered prior to commencing design and implementation. It is important that each method or approach to the solution be targeted at achieving the same set of requirements. This allows a fair judgement to be made as to which method or approach is preferable. It is wise to research existing software products as often an existing product will be a more cost effective solution. Today many software products can be customised to tailor their functionality to suit different user needs. For instance open source products can be altered to include new functionality or existing functions can be customised. Specialised functionality, such as artificial intelligence, can be outsourced or existing code modules can be obtained.

Often a method of solution is developed without consideration of other possible solutions to the problem. For example, adding another lane to a motorway may seem the obvious way to reduce traffic congestion. However, a better solution may be to encourage businesses to relocate from the city centre, or perhaps an upgrade of the public transport system should be considered. If a software solution is required there are usually a number of possible solutions to the same problem. Many consultancy services are set up for precisely this purpose. Their job is to examine different methods of approach and solutions to particular problems.



GROUP TASK Discussion

There are numerous commercial software packages which include code and even complete programs developed by other companies. Identify example software where this is the case. Explain why the development company may have decided to use code from others within their software package.

When comparing development of new software with the use of existing software solutions we should consider the following areas:

- Social and ethical considerations
- Legal issues including licensing considerations
- Customisation of existing software products
- Cost effectiveness

Social and ethical considerations

Some broad social and ethical areas that require consideration include:

- Changing nature of work for users
- Effects on level of employment
- Effects on the public

The social and ethical issues will vary depending on the problem at hand.

- **Changing nature of work for users**

The introduction of most new software products has an effect on the nature of work for users of the system. We must consider the effect the new system will have on the users. Input from potential users is critical at this stage. Ultimately, these are the people who will spend the most time with the completed product so their input is of untold value.

If the product is to be implemented across large numbers of sites, perhaps industry associations and/or unions need to be involved in the consultation process. Often, employees will be on contracts that list the tasks expected of them, and often these contracts will require re-negotiation due to changes in the nature of employees' work. Employees may need to be retrained to acquire skills for their new or changed position.

- **Effects on level of employment**

In the 1970s, a major concern was that computers were going to substantially replace the work done by many people. In many ways this has occurred, though in general computers have created many more positions that have replaced many labour intensive and repetitious jobs. One of the main reasons for creating a new software system is to reduce a business' costs. One of the largest costs for most businesses is salaries and wages, hence many new software products are designed to reduce the overall level of employment in a particular area. Developers and their clients must take this into account, both in terms of cost savings but also in terms of the effect on employees made redundant.

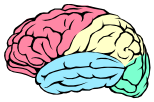
- **Effects on the public**

Large software systems can have a substantial effect on the general public. The introduction of automatic teller machines (ATMs), involved retraining the entire population. Many older people were reluctant to make use of this new banking system. The internet has changed the way many of us communicate and obtain new knowledge. Computerised billing systems have allowed us to pay bills without ever having to enter the premises of the billing company.



Fig 3.3
The introduction of ATMs involved retraining the entire population

Effects on the public can be both negative and positive. It is vital that consideration be given to how a system will affect the public during this initial stage.



Consider the following:

During the early 1980s most large supermarkets changed from a manual pricing system to a computerised system using the barcodes on individual items. More recently supermarkets and other retailers have installed self-checkout systems where only a single employee is required to monitor 10 or more checkout stations.



GROUP TASK Discussion
What effect have these changes had for supermarket employees? Explain your answers.



GROUP TASK Discussion
How has the new pricing and checkout systems affected shoppers? What measures have been introduced by supermarkets to deal with these problems?

Legal issues including licensing considerations

All parties must consider issues related to copyright (see chapter 1), both in terms of protecting the rights of the developer and the legal rights of the customers who will purchase and use the product. Software that is used to store and access private and personal information will need to include safeguards against unauthorised access.

When considering the use of an existing software solution it is worth taking the time to contact other users of the product, either directly or via online user groups. Consider also the level of support provided by the developer and whether it is possible to request new functions. The terms and conditions of the licence agreement should be read and understood. Open source licences, although free of direct costs usually do not provide direct support. Some open source products are also distributed commercially in which case professional support and training is routinely available. For businesses open source licences are not always the most cost effective solution. The extra effort required to support and troubleshoot any issues can tip the balance in favour of commercial products.



GROUP TASK Debate
"Surely the use free of open source software will always be a cheaper alternative to commercial solutions."
Do you agree? Debate using examples to support both sides.

Customisation of existing software products

Customisation of an existing software product is often a cost effective strategy for obtaining new functionality. Many software developers spend much of their time modifying their own existing products to suit the specific needs of individual clients. For example, Parramatta Education Centre writes custom database applications often using Microsoft Access to build the data entry forms and printed reports. Much of the routine work performed by the company involves upgrading existing products to include additional features. In this case Microsoft Access is being used to create and update different COTS (Customised Off-the-shelf) products.

Open source software is also routinely customised to add new features. For example, phpBB is a popular open source product used to build online forums. Many of the forums based on phpBB have been customised to suit the unique requirements of particular forums. In many cases the modifications are built as add-ons which are then made available to other users of the base software product. For example, an add-on for phpBB awards points to users based on the number and quality of posts they make. Other add-ons allow paid advertisements relevant to the subforum being viewed to be embedded within screens.

The ability to customise software helps the original developer of the product widen their market. It is now common for tools to be included in many commercial products which allow the end user to create customised screens and other functionality through the use of wizards and various drag and drop design screens. Other companies who specialise in software for particular industries offer their own software customisation service. For example, GeoCivil is a software product designed for use by civil engineers. The developers (Geocomp Consulting) offer a customisation service where GeoCivil can be altered to suit particular requirements.



GROUP TASK Research
Using the internet (or otherwise) research examples of software which can be customised. Will customising existing software products always be a more cost effective solution compared to developing a new product?

Cost effectiveness

Usually one of the constraints of a new software system will be that it falls within a certain budget. If it is not a requirement then it will most likely be an absolute constraint. This area aims to accurately assess and predict the costs involved in the development of a new software project compared with using or modifying an existing product.

Some areas of costing that should be considered include:

- Development hardware costs? Any new hardware that needs to be purchased/leased to enable development.
- Development software costs? What software is required to develop the new system? Possible purchases may include: programming languages, CASE tools, database management systems, graphics creation tools, etc.
- Development personnel costs? Essentially the salaries of the people on the development team. Also consider support staff, trainers, management and office staff.
- Outsourcing costs? Will parts of the project be outsourced to other businesses? What are the costs involved?

Once estimates and quotations have been obtained, an overall development costing can be compiled. This total development cost is then compared to the allocated budget for the project and the total cost is assessed.

How is the budget for the project obtained? Often clients for whom the project is developed will provide a budget. A budget for a project is the amount of money that is allocated for the development project. Some factors that need to be considered when formulating a budget include:

- The available capital.
- The predicted sales for the finished product.
- Cost savings as a result of the product's implementation.
- Predicted future upgrade and maintenance costs for the product.
- Ongoing customer support costs.
- Other costs. For example, special hardware required to operate the system, training costs for new future users.

It is very common for large projects to go over budget during their development. This puts management in a difficult position: a large amount of money has been spent and to scrap the project would result in large losses. To help overcome this eventuality, many budgets will include extra funds for unforeseen circumstances. It is common to add some 20% to initial costings for unforeseen circumstances. Some large projects have been known to go over budget by as much as 200%, this being particularly the case when developing new products that use new untested technologies. The American 'Star Wars' defence system is an example of a project that went way over budget and finally did not realise its stated objectives.

Recommendations resulting from questions of cost effectiveness will often involve redefining the requirements of the systems so development can proceed within budget. Perhaps some aspects of the product will be earmarked for inclusion in future upgrades once the product has an established market.



Consider the following:

A local council wishes ratepayers to be able to pay their bills on the internet. The council advertises for expressions of interest from software developers. The council has limited funds to allocate to the project and has already decided on a budget for the implementation of the system. An information pack from the council outlines the requirements of the system. As a software developer, you are interested in tendering for this job. You are currently considering applying to develop this product.



GROUP TASK Discussion
Develop a list of questions that need to be answered to determine whether the software can be developed using the available budget.



GROUP TASK Discussion
Suggest some criteria the council may use to decide which software developer should be given the contract to develop this product.

SELECTING AN APPROPRIATE DEVELOPMENT APPROACH

If an appropriate existing solution cannot be found then new software will need to be developed. This may involve developing the entire solution or it may involve writing code to customise an existing solution. When new software is developed one of the first tasks is to identify a suitable software development approach. Chapter 2 of this text describes a variety of software development approaches together with their strengths and weaknesses. In many cases a combination of approaches will be appropriate.



HSC style question:

For each of the following scenarios, recommend and justify a suitable software development approach (or combination of approaches).

(a) A breeder of an exotic type of cat wishes to record details of all cats and their kittens so they can create a family tree for each kitten they sell.

(b) A gift store wishes to commence trading over the internet. The owner has some design ideas for the website based on their current marketing, however they have concerns about protecting financial transactions performed over the internet.

(c) The manufacturer of a new printer requires a USB driver to be written that makes use of all the printer's additional features not available in standard USB drivers.

Suggested solutions

(a) If the cat breeder is computer literate and able to locate and use proprietary software, it is likely they will be able to locate a suitable package through the internet that meets her needs exactly. There are plenty of 'family tree' packages that she could use for her purposes. The approach is therefore use of an application package that she may be able to customise to her specific needs. This would require use of an existing package together with the end-user approach to customise the product to meet her specific requirements.

Another possible approach would be for her to utilise an integrated database management system, such as Microsoft Access to store the details – this would also constitute an end-user approach. If she is not capable of defining and correctly implementing the required relationships, she could ask an IT person to develop the necessary application for her using the forms and coding language of the package, thus constituting a RAD approach.

(b) The recommended approach here would be a prototyping approach to ensure that the web interface meets the client's needs exactly. The final interface designs would be developed in an iterative process using the owner's initial design ideas, with the developer progressively refining the design to incorporate both the processing requirements and effective design skills. The iterative process requires that the owner and developer work together on the design, with the designer listening carefully to the feedback from the owner at each point to ensure that the final agreed design reflects their requirements exactly.

The concerns about security can be addressed by the developer incorporating a set of routines that ensure appropriate security. These routines could either be developed specifically for the gift store website (probably using a structured approach) or more likely, the developer would make use of existing encryption routines or services which are well documented and tested.

(c) The logic required for this driver is very specific and the requirements can be defined precisely in advance. A structured approach is therefore suitable. Because this software will be distributed with new printers, it is very important that the solution be a rigorous solution, designed with the use of very formal stages. The result being a well designed, efficient, fully tested and documented system.

Correct answers could conceivably recommend a variety of different approaches. Marks would largely be awarded for justifying the chosen approach for the given scenario.

SET 3A

1. Needs can be described as:
   (A) an instance where some necessity or want exists.
   (B) what you are aiming to achieve.
   (C) the limits of a system.
   (D) factors with which the system must comply.

2. Why should requirements be verifiable?
   (A) To ensure the project remains within budget.
   (B) So the level of success or failure of the project can be evaluated.
   (C) So developers and clients reach agreement with regard to what the system will do.
   (D) To select the best approach to solving the problem.

3. Which of the following is an example of a performance issue?
   (A) Problems when the operating system is upgraded.
   (B) Features the new system must have are missing.
   (C) Poor communication between developers and clients.
   (D) Slow response to user inputs.

4. Which of the following is an example of a functionality issue?
   (A) Problems when the operating system is upgraded.
   (B) Features the new system must have are missing.
   (C) Poor communication between developers and clients.
   (D) Slow response to user inputs.

5. Which of the following is an example of a compatibility issue?
   (A) Problems when the operating system is upgraded.
   (B) Features the new system must have are missing.
   (C) Poor communication between developers and clients.
   (D) Slow response to user inputs.

6. Boundaries can be described as:
   (A) an instance where some necessity or want exists.
   (B) what you are aiming to achieve.
   (C) the limits of a system.
   (D) factors with which the system must comply.

7. Items within the boundary, outside the boundary, and communicating across the boundary. Which of the following lists these in the correct sequence?
   (A) system, interface, environment.
   (B) interface, system, environment.
   (C) system, environment, interface.
   (D) environment, interface, system.

8. Which of the following is true with regard to developing new software compared to using an existing solution?
   (A) Developing new software is cheaper than purchasing existing software.
   (B) Developing new software is always preferable to using existing software.
   (C) In many cases developing new software will better meet the unique needs of the client.
   (D) Often existing software can be used at no cost to the client.

9. Consideration of development costs in regard to hardware, software, personnel and outsourcing is part of establishing:
   (A) cost effectiveness.
   (B) functionality requirements.
   (C) system boundaries.
   (D) customer needs.

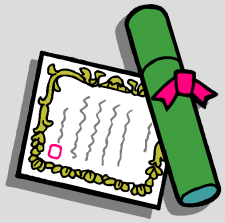
10. Which of the following types of software generally lacks personalised customer support?
    (A) Commercial software.
    (B) Customised off-the-shelf software.
    (C) Commercially distributed open source software.
    (D) Free open source software.

11. Imagine your Software Design and Development class as a system. It contains physical components such as windows, a door, lights, the computers, tables and chairs. Students in the class and the teacher, this textbook, lighting, sunlight, electricity, network connections, certain times during the week when the class is timetabled, all these things contribute to the system.
    (a) List all the items that are within the control of the class system.
    (b) List all the items that are in the environment of this system.
    (c) List any items that allow the class system to interface with its environment.
    (d) Describe the boundaries of this system.

12. Defining the problem precisely upfront is often thought to be a waste of time and resources. Do you agree? Give reasons for your answer.

13. The water board currently reads water meters manually. An employee walks from house to house recording the meter reading in a notebook. These readings are later entered into a computer at the office. Because the meter readers do not have access to previous readings, they are unable to determine abnormal usage for any individual meter. If they had access to the previous reading they could determine major water leakages as well as properties that have apparently used no water. This information would allow the meter reader to check the operation of the meter whilst on the site.
    A new computerised system is being contemplated.
    (a) What are the needs from the water board's point of view?
    (b) Transform these needs into more concrete possible functionality requirements for a new meter reading system.

14. Currently students completing the HSC in NSW receive their ATAR in December and offers of university places soon after. Many students would like to have an idea of the ATAR result they are likely to achieve after they complete their Trial HSC exams. The main purpose of this 'Trial HSC ATAR' would be to assist with decisions in regard to university. Students would also like to know any courses where extra work would best help in maximising their final ATAR.



    (a) What are the needs of the students?
    (b) Assume you are considering creating a software package to meet these needs. What would be the requirements for such a system?
    (c) Outline any social and ethical issues that would arise as a result of such a system.
    (Note: Parramatta Education Centre markets a product, 'The ATAR Estimator', available only to schools, that performs these functions).

15. A group of your friends have approached you with an idea for a new software product. This product will essentially be an integrated student information system. It will incorporate a student diary/homework system, a mark keeping system and a note taking system. You are currently considering the possibility of developing this product.
    (a) Is this problem worth solving? Describe advantages and disadvantages this new system would have compared to existing, traditional pen and paper techniques.
    (b) Do other existing software solutions already exist that solve this problem? Research to determine if similar products already exist.

DESIGN SPECIFICATIONS

Developing a set of design specifications is one of the most important steps before the actual design is planned. The design specifications form the basis for planning and designing the solution. The aim of the design specifications is to accurately interpret the needs, requirements and boundaries identified into a set of workable and realistic specifications from which a final solution can be created. These design specifications should include considerations from both the developer's perspective and the user's perspective.

Developers' perspective	**Users' perspective**
Consideration of: • Data types • Data structures • Algorithms • Variables • Software design approach • Quality assurance • Modelling the system • Documentation	Consideration of: • Interface design • Appropriate messages • Appropriate icons • Relevant data formats for display • Ergonomic issues • Relevance to the user's environment and computer configuration • Social and ethical issues

Evaluation throughout the development process is undertaken with reference to these specifications. Specifications relevant to the user's perspective can be evaluated by the user, whereas specifications relevant to the developer can be evaluated by the developer.

DESIGN SPECIFICATIONS FROM THE DEVELOPER'S PERSPECTIVE

These specifications will create a standard framework under which the team of developers must work. These are specifications that will not directly affect the product from the user's perspective but will provide a framework in which the team of developers will operate. The methods used to model the system - for example, use of system flowcharts, dataflow diagrams, IPO diagrams - will be specified. The method and depth of algorithm description to be used, how data structures, data types and variable names are to be allocated and any naming conventions that should be used should all be specified. A system for maintaining an accurate data dictionary needs to be specified. In other words, a framework for the organisation of the development process is set up so that each member of the development team will be creating sections of the solution that look, feel and are documented using a common approach (and of course so that each module will operate correctly with other modules). Often CASE tools will be utilised to ensure the team complies with these design specifications.

Once these specifications have been developed a system model can be created to give an overview of the entire system. These system models will lead to the allocation of specific tasks for completion by team members, whilst at the same time an overall direction of the project can be visualised.



GROUP TASK Discussion
What are each of the items listed under 'Developers' perspective' above? Why is it important to specify details of each of these items?

DESIGN SPECIFICATIONS FROM THE USER'S PERSPECTIVE

Specifications developed from the user's point of view should include any design specifications that influence the experience of the end-user. Standards for interface (or screen) design will be specified to ensure continuity of design across the project's screens. For example, use of menus, use of frames to group items, use of colour, placement of common elements. The wording of messages, design of icons and the format of any data presented to the user need to be determined and a set of specifications created. For example, the font to be used for prompts and user inputs, the size and nature of icons used, the tone of the language used.

Ergonomic issues should also be considered and taken into account as part of the design specifications. Questions such as: Which functions will be accessed most often? Which functions require keyboard shortcuts? How should movement between screen elements be accomplished? What is the order in which data will be entered? Are the screens aesthetically pleasing? Such questions need to be examined and a standard set of design specifications generated.

The user's environment will also have an effect on the specifications created. Perhaps the users are familiar with existing applications. If so, these applications should be examined and some of their design elements incorporated so that transfer of skills from existing applications can take place. For example, using keyboard short cuts that are already known from other applications. Operating system settings should be used to determine the following: colour of windows, typefaces used and printer settings. Consideration of hardware necessary to execute the new product including: processor speed, RAM, video memory and hard disk space. Potential users need to be consulted to determine the acceptable specifications.

System models can assist in determining user based design specifications, in particular, screen designs and concept prototypes. It is vital that software developers acknowledge the contribution users make to the development of design specifications. Communication and in particular feedback from users is especially important at this early stage.



Consider the following:

A franchise based lawn mowing company wishes to have a software product developed to ensure all its franchisees quote and invoice their clients using a standard approach. The software needs to run on very basic MS-Windows machines with very limited RAM. The feasibility study has shown that the computer skills of many of the franchisees are very limited to non-existent.



GROUP TASK Discussion
List and describe some of the design specifications from the user's perspective that need to be incorporated into this project.



GROUP TASK Discussion
Concept prototypes are one method for gathering user specifications, however the limited computer skills of the users may limit the usefulness of such prototypes. Brainstorm and discuss other possible strategies that could be used to gather user specifications.

COMMUNICATION ISSUES

In the previous section, we examined the need to consider the user's perspective in terms of developing design specifications for software projects. The users of the final product are ultimately the software developer's clients. Their needs should be considered at all stages of the software development cycle and especially whilst defining and understanding the problem. It is vital to open the communication channels between users and the software development team. Communication is two-way: lines of communication should be created that empower the users and enable them to provide feedback to the development team. In addition, when users are consulted and their feedback is incorporated into the product's design then many training and future support issues will be avoided.

Communication with users could include:

- Meetings
- Questionnaires
- Memos
- Interviews
- Phone calls and Emails
- Concept prototypes

The methods used should be a mix of formal and informal communication. Formal meetings will be required to accomplish specific tasks, however informal communications will often elicit more useful feedback about both the existing system and perceived problems with the new system.

Users should be kept informed about the progress of the new system as much as is possible. Often people will be concerned about the effects of a new system on their own work. Questions such as: Will I be able to understand the new system? Will the new system mean more work for me? Why change something that works? are common when changes are made to people's work practices. Change is easier to accept and deal with when you are fully informed. If users have been involved in the development process they will feel some ownership of the final product. Ownership builds confidence, and user confidence in the new system results in a smoother and more successful implementation.



GROUP TASK Discussion
Often the users feel alienated from the design of new systems. Discuss methods software developers could use to enable and accept feedback from users.



GROUP TASK Debate
Software developers should be more interested in solving the problem than in creating elegant screens for users. Do you agree with this statement? Debate each side.

SET 3B

1. Design Specifications can best be described as:
   (A) accurately identified needs, requirments and boundaries.
   (B) the user's evaluation of the system.
   (C) the developer's evaluation of the system.
   (D) a framework for how the system will be developed.

2. DFDs and IPO diagrams are both examples of:
   (A) design specifications.
   (B) CASE tools.
   (C) methods used to model a system.
   (D) data types.

3. Ergonomic issues can be described as a consideration of:
   (A) the developer's perspective.
   (B) the user's perspective.
   (C) the social perspective.
   (D) the system perspective.

4. System models are useful because they can assist design teams in:
   (A) developing the design specifications.
   (B) allocating specific tasks that need to be completed.
   (C) maintaining an accurate data dictionary.
   (D) creating a standard framework for the project.

5. Identify an example of informal communication from the list below:
   (A) scheduled meeting.
   (B) questionnaire.
   (C) internal company memorandum.
   (D) general discussion in the canteen during lunch break.

6. To ensure that design specifications are adhered to a team of developers will often use:
   (A) CASE tools.
   (B) IPO diagrams.
   (C) flowcharts.
   (D) formal and informal communication methods.

7. Ergonomic issues to be considered by the developer include:
   (A) aesthetically pleasing screen designs.
   (B) keyboard shortcut functionality.
   (C) the order in which users input data.
   (D) all of the above.

8. A software solution that does not cater to the needs of a physically challenged user lacks consideration in which area?
   (A) social and ethical.
   (B) technical.
   (C) environmental.
   (D) documentation.

9. It is important for developers to communicate with the users because:
   (A) users are often scared of change.
   (B) users will feel more confident about the new system.
   (C) users can provide useful feedback on the development of the system.
   (D) all of the above.

10. When considering the environment of the user, the developers should make themselves aware of:
    (A) current hardware configuration being used.
    (B) current applications being used.
    (C) current operating system settings.
    (D) all of the above.

11. Communication in all forms between the developers and the users is critical to the successful development and implementation of a software solution. In your own words describe different methods of communication that should be employed to ensure this happens and explain the potential resulting outcomes if communication does not take place.

12. Outline areas that should be considered when developing a set of design specifications.

13. The same ergonomic design issues arise regardless of the industry or the software solution that is under development. Do you agree with this statement? Give examples that justify your answer.

14. The same social and ethical design issues arise regardless of the industry or the software solution that is being developed. Is the above statement true? Give examples that justify your answer.

15. You are a member of a software design team that has been employed to develop a solution that will be used by people who are visually impaired. Some of the users have only 10% sight. Describe some of the features that you could include in the design specifications that would meet the needs of all end-users of the new system.

SYSTEM DOCUMENTATION

A variety of different types of documentation are produced throughout the software development cycle. The development team should decide in advance which particular set of documentation tools they will utilise to assist with the development process. There are numerous different methods for modelling different aspects of the software engineering process. Today many larger software companies utilise diagrams that form part of the UML (unified modelling language) which incorporates a variety of different modelling tools that are now available as part of many CASE tools and IDEs (integrated development environments).

In software design and development we restrict our study to the following set of systems modelling tools:

- System flowcharts
- Context diagrams
- Data flow diagrams (DFDs)
- Structure charts
- IPO diagrams
- Data dictionaries
- Screen designs (and concept prototypes)
- Storyboards

Other significant documentation produced during the development process includes:

- Algorithms represented using pseudocode or flowcharts to document the logic in modules and subroutines (chapter 4).
- Test data together with expected outputs (chapter 4, 5 and 6).
- Source code including internal comments and intrinsic documentation (chapter 5).

SYSTEM MODELLING TOOLS

A model of a system is a representation of that system designed to show the structure and functionality of the system. Many system modelling tools are in the form of diagrams. Diagrams are particularly useful methods of modelling because they are able to give a broad view whilst at the same time conveying necessary detail. Accomplishing the same task in words is difficult. In terms of software development, a model can be thought of as a plan. The model gives direction and specifications to the builders of the product, in the same way as the plan for a house gives builders of the house direction and specification in regard to the house's erection.

Different types of modelling are applicable to different aspects of the system. System flowcharts are used to represent the logic and movement of data between the system's components, including hardware, software and manual components. Dataflow diagrams describe the flow of data to and from processes and storage elements. Structure charts describe the top-down design and sequence of processing. Storyboards describe the navigation between screens. IPO diagrams explain how inputs are transformed into outputs by processing. Data dictionaries describe the nature and type of the data used in a program. Screen designs and concept prototypes are used to determine user requirements by simulating the final product from the user's perspective. Most projects will use a combination of modelling techniques.

SYSTEM FLOWCHARTS

System flowcharts are used to describe the logic and flow of data through a system. They describe the interactions that occur between input, processing, output and storage, as well as the nature of each of these components. Even manual processes are included on system flowcharts. Generally, system flowcharts are used at a higher level than the other modelling techniques to show an overall view of the entire system.

Although some of the symbols used on system flowcharts are the same as in algorithm flowcharts, the design of system flowcharts is quite different. Flowcharts for algorithms are designed to precisely show the logic of an algorithm, whereas system flowcharts are not describing an algorithm but a system. Logic is only shown where necessary to sensibly describe the flow between components.

The following symbols are used on system flowcharts:

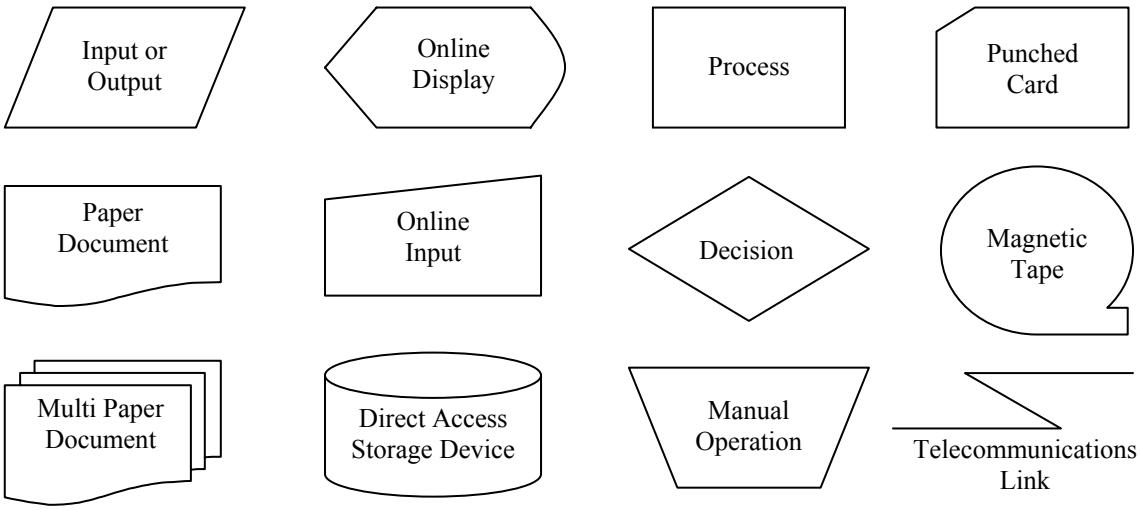


Fig 3.4
Components of System Flowcharts.



Consider the following system flowchart:

The system flowchart in *Fig 3.5* is a basic outline of a program for maintaining an inventory file of all the products currently in stock within a warehouse. As items are sent out as part of customer orders the number of items in the inventory is reduced. If an order contains items not currently in the inventory then they must be backordered.

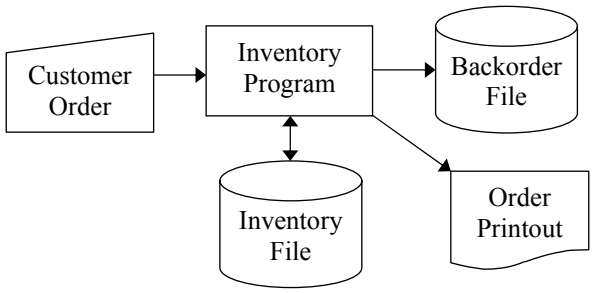


Fig 3.5
System flowchart for an inventory program

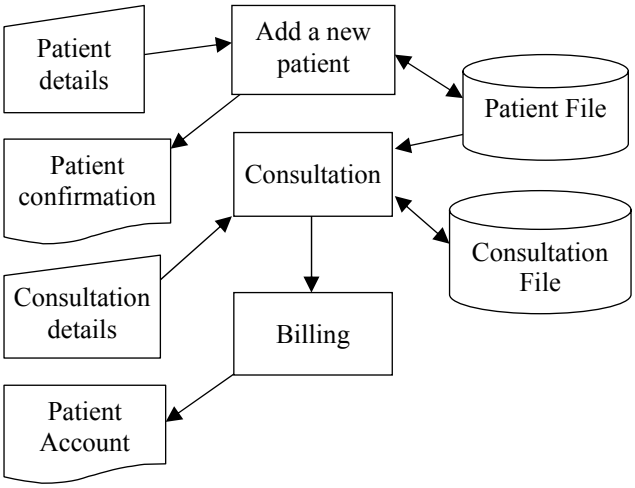


GROUP TASK Activity
Expand the system flowchart in *Fig 3.5* so it includes the logic described above which is performed by the inventory program.



HSC style question:

The following System Flowchart represents part of a new system proposal to be used by a local doctor to manage their patient's information.



(a) The above system flowchart includes four different symbols. What is the meaning of each of these four symbols?

(b) Describe the system as modelled in the system flowchart above.

(c) Modify the system flowchart to include a process (together with its inputs and outputs) which allows the doctor to view all the consultation details for a given patient on the screen.

(Answer (c) by modifying the systems flowchart above)

(d) During the first few meetings held between the Systems Analyst and the Doctor, it quickly became apparent that there were some initial difficulties due to the different perspectives of the system as viewed by the Analyst and Doctor.

Explain why these difficulties probably arose, and how they would have affected the communication process between the Analyst and the Doctor.

Suggested solutions

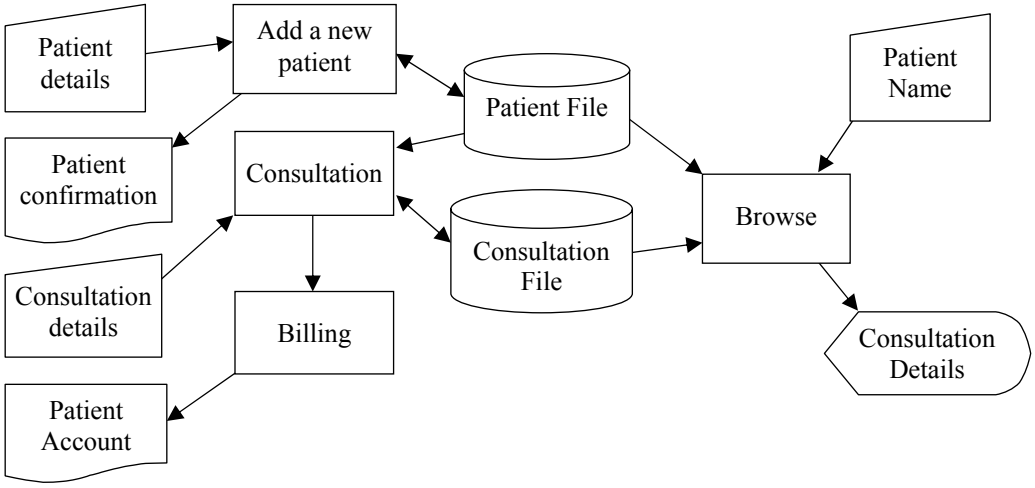
(a) The symbol  indicates online input usually via a keyboard, such as a user entering patient details.  represents a process performed by the system, such as adding a new patient to the system.  represents a paper document and/or printer which produces a hardcopy report such as the patient confirmation printout. The symbol  is a disk drive or file used to store data, such as the patient file which holds records of patient details.

(b) When a new patient is added, the patient details are entered through the keyboard (presumably by the doctor or receptionist). A new patient record is written out to the patient file and then a patient confirmation report is printed which presumably is handed to the patient so they can verify the details are correct.

When a consultation occurs, details of the consultation are entered which presumably includes sufficient data so that the patient's record can be read from

the patient file. The consultation details are recorded as consultation record which is written to the consultation file. The Billing process is then executed, which produces a hard copy of the account for the patient.

(c)



(d) Differences are likely due to the developer focussing on how they will implement the solution. For instance concentrating on data structures, file structures, and algorithms. The use of 2 separate files for Patient file and Consultation file and the particular structures and algorithms that will be used may well be a good way to develop the functionality required by the doctor, however the doctor is not interested in this detail.

The Doctor focuses on the features he wants to be able to use in his practice, and how the interfaces will look, without any knowledge (or even interest) in how this will be achieved by the developer. He will be concerned about the use of his company logo, the way the consultation details are displayed, patient confidentiality, how a new patient is entered into the system, and whether the system will run in his rooms on his existing equipment.

Communication will be hampered initially if the developer does not clearly listen to the needs of the doctor and the required functionality of the proposed system, without being distracted by what the developer can offer technically. The use of a prototype would help to overcome these initial difficulties by ensuring that the doctor's needs are heard and demonstrated practically.



GROUP TASK Discussion
The system flowchart does not detail how consultations are linked to individual patients. Discuss strategies that could be used to solve this problem.



GROUP TASK Discussion
There are various issues suggested by the current system flowchart. Identify these issues and suggest possible solutions.

CONTEXT DIAGRAMS

Context diagrams represent the entire system as a single process. They do not attempt to describe the processes within the system; rather they identify the data entering and the information leaving the system together with its source and its destination (sink). The sources and sinks are called "external entities". As is implied by the word external, these entities are present within the system's environment. Context diagrams are really top-level data flow diagrams and are often known as level 0 data flow diagrams.

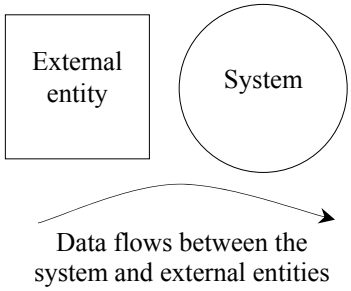


Fig 3.6
Symbols used on context diagrams.

Squares are used to represent each of the external entities. Common examples of external entities include users, other organisations and other systems. These entities are not part of the system being described, as they do not perform any of the system's processes. Rather the system acquires (inputs) data from each source entity and/or the system supplies (outputs) information to each sink entity. The entire system is represented using a circle, with labelled data flow arrows used to describe the data and its direction of flow between the system and its external entities. Data flows from each source into the system, and data (information) flows from the system to each sink. In some references, external entities are also known as terminators as they describe starting and ending points for the DFD.

Each data flow label should clearly identify the nature of the data using simple clear words. Remember each data flow describes data not a process, for example if a user enters a password then an appropriate data flow label would be "User password", not "Enter password". Furthermore in this example each user is the source of a single password, so "User password" is a more appropriate label than "User passwords". If many data items flow together then a plural label would be more appropriate, however in most systems this is a rare occurrence.

All data entering the system and all data (information) leaving the system must be included on the context diagram. All processes performed by the system and data stored by the system are part of the single system circle and are not detailed on the context diagram. Consider the context diagram in *Fig 3.7*. One of the sales team

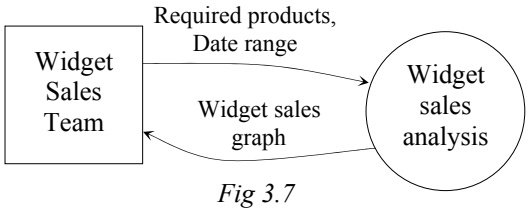


Fig 3.7
Widget sales analysis context diagram.



Fig 3.8
Alternate Widget sales analysis context diagram.

enters a list of required products and a date range and the system responds by outputting a widget sales graph. Clearly this system is storing sales data and is performing a variety of processes using this data to generate the sales graph. *Fig 3.8* shows an alternate version of the Widget sales analysis context diagram to illustrate how external entities can be included multiple times. Both versions are logically identical.

So how does a context diagram assist when understanding the problem? Context diagrams indicate where the new system interfaces with its environment. They define the data and information that passes through each interface and in which direction it travels. Descriptions of this data and information is further detailed within a data dictionary. Ultimately the data entering the system from all its sources must be sufficient to create all the information leaving the system to its sinks.

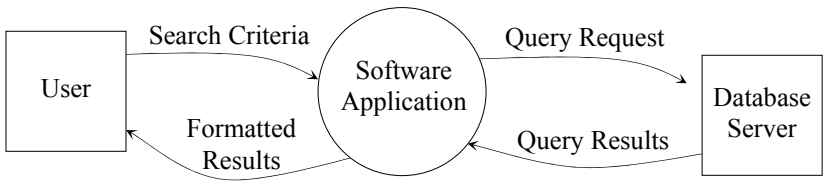


HSC style question:

A software application uses search criteria entered by the user and then creates and sends a query request to an external database server. The database server responds with the query results. The software application then formats the query results and displays the formatted results on the user's screen.

Construct a context diagram for this software application.

Suggested solution



DATA FLOW DIAGRAMS (DFDs)

Data flow diagrams describe the path data takes through a system. No attempt is made to indicate the timing of events. Think of a DFD as a railroad map: it shows where the train tracks are laid, but it does not give the timetables.

DFDs do not attempt to describe the step-by-step logic of individual processes within a system. Rather they describe the movement and changes in data between processes. As all processes alter data then the data leaving or output from a process must be different in some way to the data that entered or was input into that process. This is what all processes do; they alter data in some way. The aim of DFDs is to represent systems by describing the changes in data as it passes through processes. For example a process that adds up numbers receives various numbers as its input and outputs their sum. On DFDs there is no attempt to describe how the numbers are summed. Rather the emphasis is on where the numbers come from and where the sum is headed.

To represent the data moving between processes we use labelled data flow arrows. The label describes the data and the direction of the arrow describes the movement. Processes are represented using circles. The label within the circle describes the process. As processes change data the labels used should imply some action – verbs should be used, such as create, update, collect, to emphasise that some action is performed.

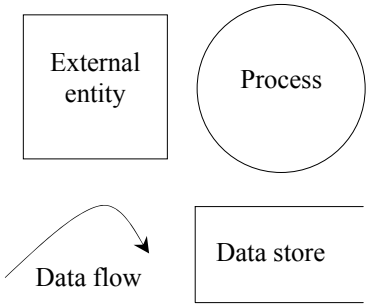


Fig 3.9
Symbols used on data flow diagrams.

The final symbol used on DFDs represents data stores. A data store is where data is maintained prior to and after it has been processed. In most cases a data store will be a file or database stored on a secondary storage device,

however it could also be some form of non-computer storage such as a paper file within a filing cabinet. An open rectangle together with a descriptive label is used to represent data stores. Data stores allow the system to pause or halt between processes and they also allow processes to occur in different sequences and at different times. In effect processes are freed to execute independently of each other. Consider a typical process that collects data from a user and stores it within a data store. This single process can execute many times simultaneously whilst at other times it sits idle. The data is maintained within the data store where it can be retrieved and used by other processes when and as they require.

As data stores are simply collections of data they cannot perform processing, therefore all data flow arrows heading into and out of a data store must be connected to a process. A process is required to write or read data to and from a data store.

Context diagrams are top-level data flow diagrams – also called level 0 DFDs. They specify all external entities with the complete system represented as a single process. A level 1 data flow diagram expands this single process into multiple processes. A series of Level 2 DFDs are drawn to expand each level 1 DFD process into further processes. Level 3 DFDs similarly expand each level 2 process and so on. A series of progressively more and more detailed DFDs refine the system into its component sub-processes. Eventually the lowest level DFDs will contain processes that can be solved independently. Breaking down a system's processes into smaller and smaller sub-processes is known as 'top-down design'. The component sub-processes can be solved and even tested independent of other processes. Once all the sub-processes are solved and working as expected they combine to form the complete solution.

On some level 1 and lower-level DFDs the external entities are included, whilst on others they are not. If a context diagram has already been produced or external entities have been included on a higher-level DFD then it is common practice to omit the external entities from the derived lower-level DFDs. A similar practice is also true for data stores, however in the interest of improved clarity it is more common to reproduce data stores on lower-level DFDs. As was the case with context diagrams, to improve clarity it is permissible to include the same external entity multiple times and similarly a single data store can be included multiple times. For instance in *Fig 3.10* below the "Widget Sales Team" entity is included twice simply to improve readability.

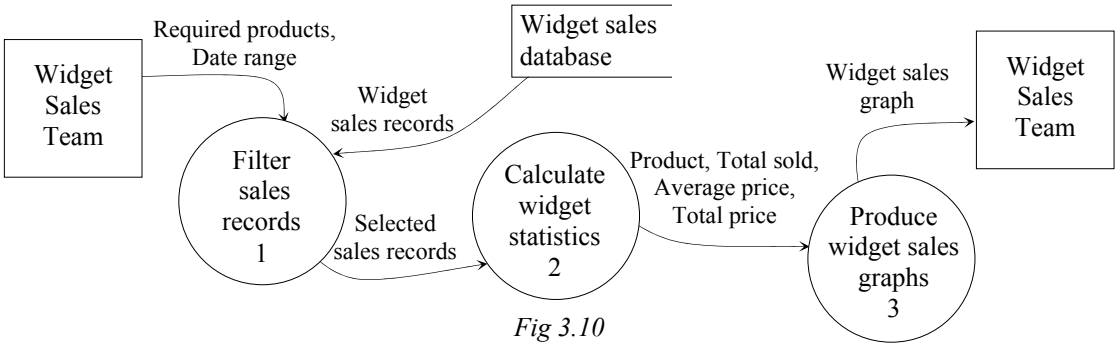


Fig 3.10
Sample Level 1 Widget data flow diagram.

On most DFDs the processes are numbered in addition to their labels. Consider the example level 1 DFD in *Fig 3.10* – it contains the three processes, 1. Filter sales records, 2. Calculate widget statistics and 3. Produce widget sales graphs. Three level 2 DFDs would then be produced – one for each process in the level 1 DFD. *Fig 3.11* shows an expansion of process 2. Calculate widget statistics into a level 2 DFD containing four processes. These four processes are numbered from 2.1 to 2.4 – the 2

indicating their connection to process 2 on the level 1 DFD. If process 2.1 required further expansion into a level 3 DFD then its processes would be numbered 2.1.1, 2.1.2, 2.1.3 and so on.

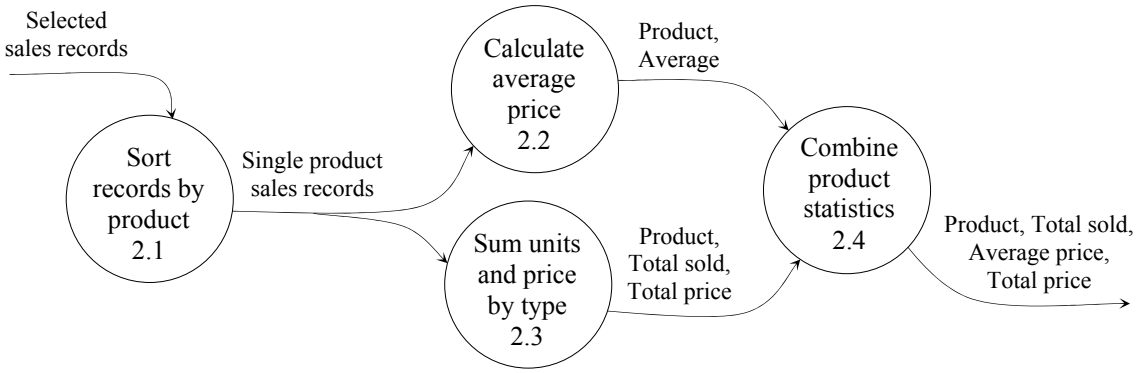


Fig 3.11
2. Calculate widget statistics DFD.

Data flow diagrams are used both as a tool for system analysis when understanding the problem and also as a tool for system design. Modelling an existing system using a data flow diagram assists in understanding the flow of information through a system and also helps the analyst to separate processing into distinct units resulting in a better understanding of the problem. Data flow diagrams are particularly useful modelling tools during system design and form part of the UML (unified modelling language). CASE tools are available that allow the production of data flow diagrams. Many of these tools include data dictionary functions that link to the data flows on the diagrams.



GROUP TASK Discussion
Based on the above Widget sales analysis system modelled in *Fig 3.9, 3.10* and *Fig 3.11* discuss a suitable format for the sales graphs.



Consider the following DFD summary points:

- All processes must have a different set of inputs and outputs.
- All lower-level DFDs must have identical inputs and outputs as the higher-level process they expand.
- External entities and data stores can be reproduced on lower-level DFDs.
- External entities must be present on context diagrams (level 0 DFDs) but are optional on lower-level DFDs.
- Data flows into and out of a data store must be connected to a process. They can never directly connect to an external entity or another data store.
- A single output data flow can be the input to multiple other processes.
- Labels for processes should include verbs that describe the action taking place.



GROUP TASK Activity
Identify examples within *Fig 3.10* and *Fig 3.11* above that illustrate each of the above dot points.

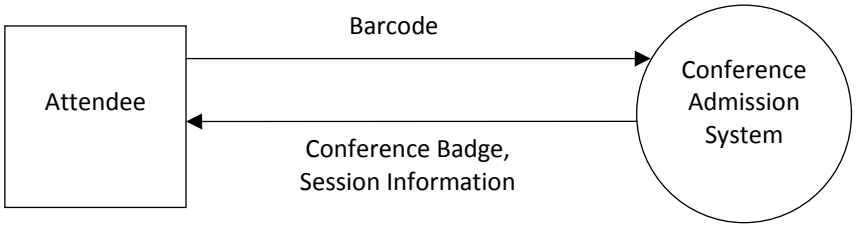


HSC style question:

An automated conference admission system has been designed to allow for the efficient registration of conference attendees as they arrive at the conference venue. The system operates according to the following guidelines:

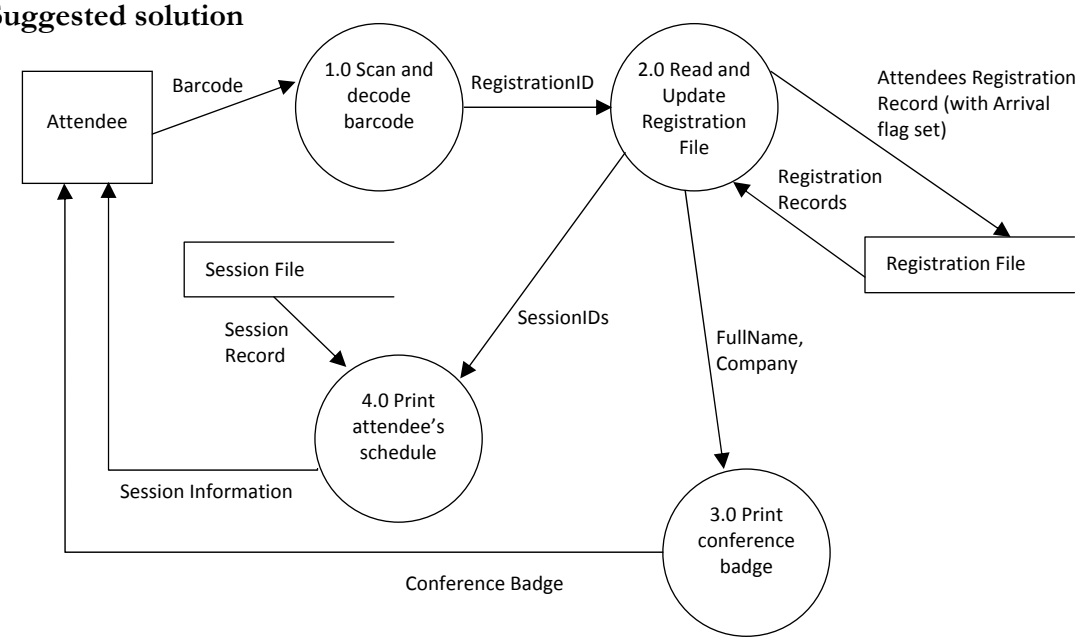
- Upon arrival at the conference centre the attendee scans the barcode on a printout (which they printed as part of their online conference registration). The barcode encodes the attendees unique RegistrationID.
- The system updates the existing Registration File to indicate that the attendee has arrived. This file contains a Registration record for each attendee, including their RegistrationID, Fullname, Company, Arrival flag and the SessionID for each session they are scheduled to attend.
- The system then prints the attendee's personalised conference badge, which includes their FullName and Company from the Registration File.
- The system also prints information about each of the attendee's scheduled sessions. The existing Session File contains a Session Record for each session. Each record contains SessionID, Time, Title, Presenter and Description.

A context diagram for the above system has been prepared.



Construct a data flow diagram for the Conference Admission System.

Suggested solution



Note the detail of the Registration Records *being read from the file. The correct record is found and used three times, namely to update the record and also to provide* FullName *and* Company *to the* Print conference badge *process and* SessionIDs *to the* Print attendee's schedule *process.*

STRUCTURE CHARTS

Structure charts (or structure diagrams) are used to model the hierarchy of subroutines (processes) within a system, together with the sequence in which these subroutines are executed. Data movements between subroutines are included to improve the reader's understanding of the relationships between them. In terms of software development, structure charts describe the top-down design of the program together with the order in which subroutines are called. Because subroutines can be conditionally called or repeatedly called from a higher-level subroutine, decisions and repetitions are also indicated on structure charts.



Call
Causing the execution of a subroutine (process) from within another subroutine.

The primary function of a structure chart is to create a template in preparation for the creation of the actual source code. Each subroutine shown on the structure chart will become a procedure or function in the final program. For large software projects, structure charts may be used to allocate tasks to individual programmers on the development team. Structure charts are also useful tools for maintaining and upgrading software. Identifying subroutines and modules that could potentially contain an error or that need upgrading is made easier if a structure chart is available. CASE tools are available that can analyse existing source code and automatically generate models similar to structure charts.

There are many recognised techniques for drawing structure chart type models and many different names are used for these diagrams. Some common names include structure diagram, function chart, call graph, call tree and hierarchy diagram. The important common theme for all these modelling techniques is that they attempt to describe the top-down, hierarchical design of a system. Some include an indication of the sequence of execution and methods for indicating decisions and repetitions.



Control
The influence that causes tasks to execute in their correct sequence.

Let us now consider the symbols used in structure charts as they are to be drawn in this course. Processes, which are usually subroutines and sometimes modules, are drawn as rectangles. Lines drawn between processes are known as call lines or control lines. They show the connections between processes. Each line represents a call to a subroutine from the subroutine above. Decisions about whether a call should be made are indicated using a small diamond. If a subroutine is to be called multiple times, then a circular arrow to indicate this repetition, is placed around the call line. Data items passed between tasks are known as parameters and are shown using a small open circle with an arrow attached to indicate the direction of the data movement along the call line. A filled in circle with attached arrow is used to show a control parameter. Control parameters have an effect on the order in which tasks are executed. Usually, a control parameter will be a flag. Flags are data items used to indicate that a certain criterion has been met.

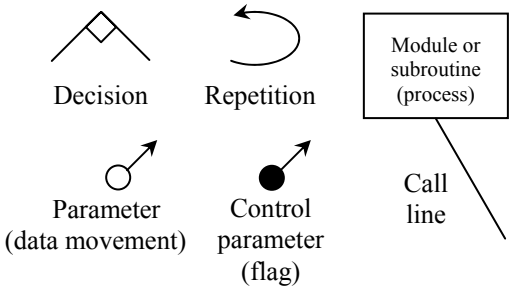


Fig 3.12
Symbols used in structure charts.

Structure charts can be read from top to bottom. Lower-levels represent subroutines that are called from the subroutine above. Reading from left to right on a particular level describes the order of execution of tasks. Sometimes a subroutine may be called by more than one higher-level subroutine, because order of execution is from left to right. It is necessary to include this subroutine each time it is called by a different higher-level task.



Consider the following structure chart and algorithms:

A program is being developed for an automotive spare parts business. The structure chart in *Fig 3.13* below describes the subroutines used to determine the number of items currently held in stock for particular parts.

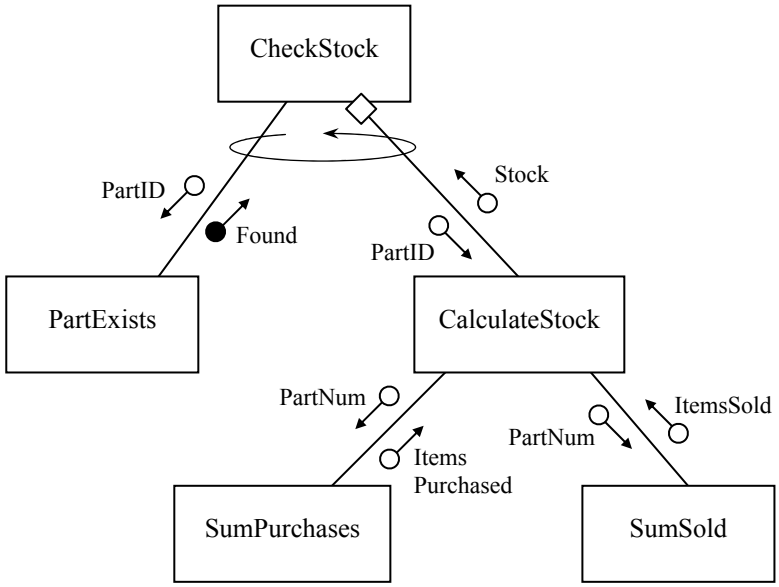


Fig 3.13
CheckStock structure chart.

The following algorithms have been written based on the structure chart in *Fig 3.13* above. Algorithms for the SumPurchases and SumSold subroutines are yet to be developed. Line numbers are included to assist with the discussion that follows.

```
1   BEGIN CheckStock
2       Get PartID
3       WHILE PartID ≠ 0
4           IF PartExists(PartID) THEN
5               Stock = CalculateStock(PartID)
6               Display Stock " item(s) in stock"
7           ELSE
8               Display "Invalid part number."
9           ENDIF
10          Get PartID
11      ENDWHILE
12  END CheckStock
```

```
13  BEGIN PartExists(PartNum)
14      Found = False
15      Open Parts file for input
16      Read first record into TempRec from Parts
17      WHILE TempRec.PartID ≠ 0 AND Found = False
18          IF TempRec.PartID = PartNum THEN
19              Found = True
20          ELSE
21              Read next record into TempRec from Parts
22          ENDIF
23      ENDWHILE
24      Close Parts file
25      RETURN Found
26  END PartExists

28  BEGIN CalculateStock(PartNum)
29      ItemsPurchased = SumPuchases(PartNum)
30      ItemsSold = SumSold(PartNum)
31      Stock = ItemsPurchased - ItemsSold
32      RETURN Stock
32  END CalculateStock
```

First consider how the structure chart represents the hierarchy or top-down design of the subroutines. In the structure chart we have a total of five subroutines. The CheckStock subroutine calls both the PartExists and the CalculateStock subroutines. The CalculateStock subroutine calls the SumPurchases and the SumSold subroutines. Notice how the top-down design is very obvious on the structure chart but is somewhat difficult to determine using the algorithms alone.

The left to right placement of the subroutines within the structure chart indicates the order in which they are called. For example, the CheckStock subroutine first calls PartExists then calls CalculateStock, similarly the CalculateStock subroutine calls SumPurchases and then calls SumSold.

On the structure chart calls from the CheckStock subroutine are surrounded by a repetition symbol ⟲. This means that within the CheckStock algorithm the two calls are made within a loop. Examining the algorithm we find the pre-test loop beginning at line 3 and ending at line 11 surrounds the PartExists and CalculateStock calls.

A decision diamond ◇ is included on the CalculateStock call line. This means the CalculateStock call is optional based on a decision within the CheckStock algorithm. Examining the CheckStock algorithm we find line 4 to 9 includes the relevant logic which avoids calling CalculateStock if the part number is not valid.

Finally consider the parameters on the structure chart. Based solely on the structure chart it is clear that the subroutine PartExists receives a PartID and returns Found. The Found parameter is represented as a control parameter ➶. This implies it is most likely a Boolean which is used to make a decision on a future call. In this case Found is used to decide if the CalculateStock subroutine needs to be called. The CalculateStock subroutine receives a PartID and returns the Stock of that part. The SumPurchases subroutine is sent a PartNum and returns the number of ItemsPurchased, similarly SumSold receives a PartNum and returns the number of

ItemsSold. As these parameters use the symbol ♂ they are not control parameters and hence they do not alter the order of processing (or flow of control) within the algorithms. Rather these parameters are used to pass data into and out of subroutines.



GROUP TASK Discussion
Structure charts can be created from existing algorithms or even from existing source code. Examine the algorithms above to determine how they convert to the initial structure chart.

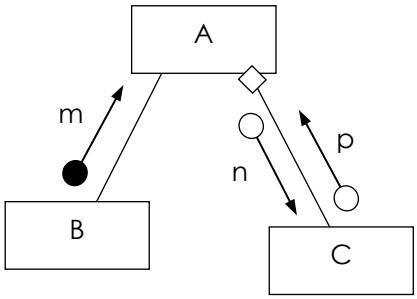


GROUP TASK Discussion
The PartID entered by the user is utilised by all the subroutines. Describe how the PartID is passed to each of the subroutines.



HSC style questions:

Consider the following structure chart when answering Question 1, 2 and 3.



1. Which of the following correctly identifies elements on the structure chart?

(A) m, n and p are processes, A, B and C are parameters.

(B) n and p are control parameters, m is a parameter, A, B and C are processes.

(C) m is a parameter, n and p are data items, A, B and C are variables.

(D) m is a control parameter, n and p are parameters, A, B and C are processes.

2. Which of the following best describes the structure chart?

(A) A calls B which returns a Boolean value for m. m is used to decide if C is called. If C is called it returns p after processing the value n.

(B) B calls A with the Boolean value m. m is used to decide if C should call A. If A is called it returns n after processing the value p.

(C) A calls B which returns a Boolean value for m. C is called repeatedly using n values. Each time C is called it returns a value for p.

(D) A calls B with a value for the parameter m. C is optionally called with the parameter n and returns a value for p.

3. Which of the following algorithms for A best reflects the above structure chart?

(A)
```
BEGIN A
    B = m
    IF m THEN
        Get p
        C = B(n)
    END IF
END A
```

(B)
```
BEGIN A
    B(m)
    C(n, p)
END A
```

(C)
```
BEGIN A
    m = B
    IF m THEN
        Get n
        p = C(n)
    ENDIF
END A
```

(D)
```
BEGIN A
    B(m)
    IF m THEN
        C(n, p)
    END IF
END A
```

Question 4.

The following algorithm updates the high score data at the conclusion of a game.

```
BEGIN UpdateHighScores(PlayerName,Score)
    LoadHighScores
    PlayerIndex = FindPlayer(PlayerName)
    IF PlayerIndex < 0 THEN
        InsertPlayer(PlayerName, Score)
    ELSE
        IF Score > Player(PlayerIndex).HighScore THEN
            UpdateScore(PlayerIndex, Score)
        ENDIF
    ENDIF
    WriteHighScores
END UpdateHighScores
```

Create a structure chart for the UpdateHighScores algorithm.

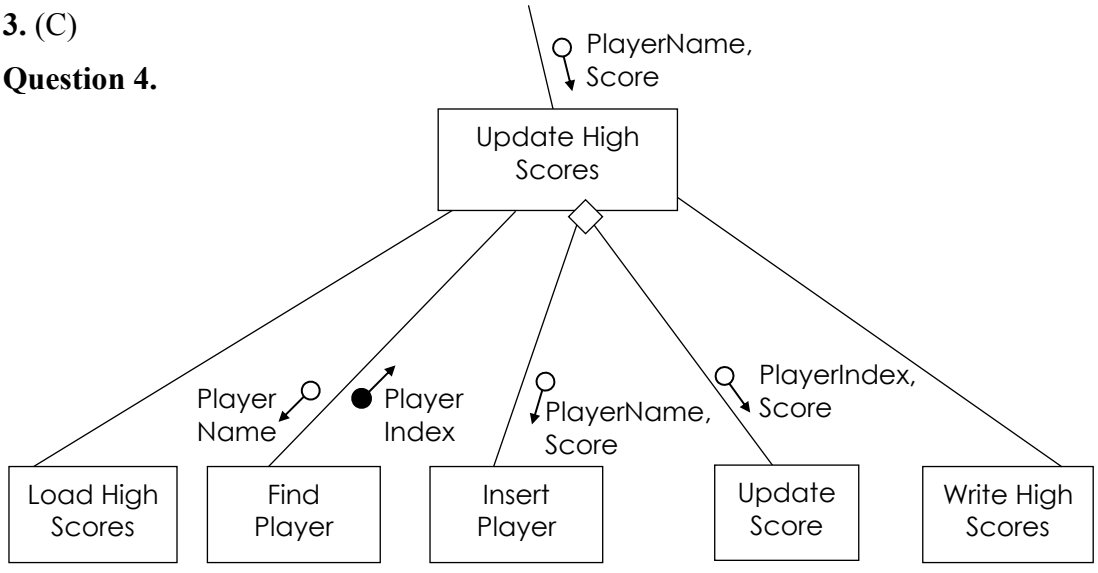
Suggested solutions

1. (D)

2. (A)

3. (C)

Question 4.



INPUT PROCESS OUTPUT (IPO) DIAGRAMS

IPO diagrams are used to describe the data entering a process, the nature of the processing performed on the data and the resulting information leaving the process. IPO diagrams are used to further expand on the lower-level processes described on data flow diagrams and structure diagrams.

There are many ways of drawing IPO diagrams or charts. An IPO diagram can be a graphical diagram used to show the flow of control from an input through a process transforming the input into an output (see *Fig 3.14)*. The resulting output will become the input for a further process or processes. In other words, the diagram shows a number of processes with the focus on the links between their outputs and inputs. In this course IPO diagrams use a modified table format, with a section for inputs into a particular process, a section for an explanation of the processing and a section to list the outputs from the processing. *Fig 3.15* shows three commonly used methods of drawing IPO diagrams. Each of the methods is conveying virtually identical information: it is just the format that is different.

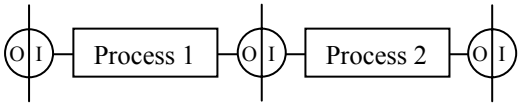


Fig 3.14
Graphical interpretation of an IPO diagram.

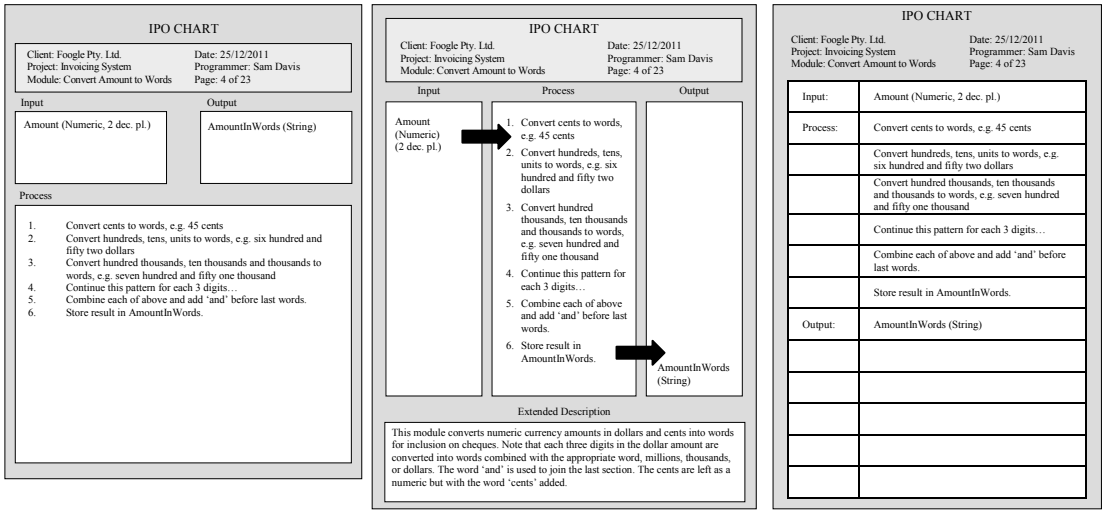


Fig 3.15
Three commonly used methods of drawing IPO diagrams or charts.

Different software development companies will use different formats for their IPO diagrams. IPO diagrams form the specifications that are handed to programmers. The programmer uses the information to write algorithms and finally the source code for the module. The process section of the IPO diagram need not explain all the logic of the solution but must include the precise nature of the tasks the subroutine or module must accomplish. In the above example, a precise explanation is given of how to convert amounts to words, however no indication is given in regard to decisions and repetitions that may be required in the final algorithm.

The recommended method for drawing IPO diagrams in this course, is to use three columns: one each for input, process and output. The inputs and the outputs are aligned next to the process that accepts the input or the process that results in the output.

Fig 3.16 shows an example of this method using the same information as was used in the previous *Fig 3.15* examples. On the IPO diagram at right, the input, Amount is shown next to the first process that requires that input. The output, AmountInWords, is shown next to the process that ultimately calculates the value of that variable. Further information in regard to the source and destination of the inputs and outputs can also be included in the input and output columns.

IPO CHART

Client: Foogle Pty. Ltd.
Project: Invoicing System
Subroutine: Convert Amount to Words

Date: 25/12/2011
Programmer: Sam Davis
Page: 4 of 23

Input	Process	Output
Amount	Convert cents to words, e.g. 45 cents	
	Convert hundreds, tens, units to words, e.g. six hundred and fifty two dollars	
	Convert hundred thousands, ten thousands and thousands to words.	
	Continue this pattern for each 3 digits…	
	Combine each of above and add 'and' before last words.	
	Store result in AmountInWords.	AmountInWords

Fig 3.16
Recommended method for drawing IPO diagrams.



Consider the following:

A business is invoiced for supplies from its vendors. The structure chart below models the tasks involved in processing and paying these invoices. A series of IPO diagrams are now being prepared.

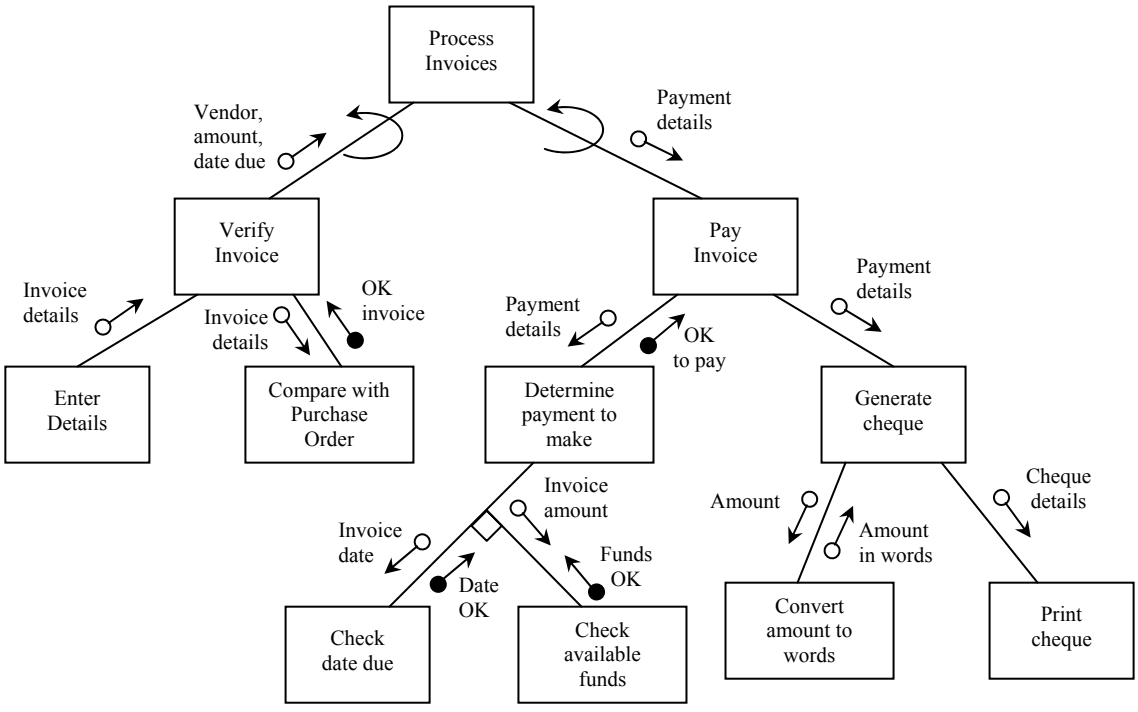


Fig 3.17
Structure chart for processing invoices.



GROUP TASK Discussion
Describe the processing modelled on the above structure chart (*Fig 3.17*). Identify each input into the module and the most likely data source.

An IPO chart has been developed for the Compare with Purchase Order task from the structure chart in *Fig 3.17* above.

IPO DIAGRAM

Client: Foogle Pty. Ltd.
Project: Invoicing System
Subroutine: Compare with Purchase Order

Date: 29/12/2011
Programmer: Janine Fendall
Page: 9 of 23

Input	Process	Output
	Set OKInvoice to True	
PO Number, Invoice Item Name, Cost, Quantity (from VerifyInvoice)	Read PO details from PO file using PO number.	
Purchase order Item Name, Cost, Quantity (read from PO file)	Check if Item Names match and Item Costs match and Item Quantities match.	If False then OKInvoice is set to False
	Add Invoice Item Cost to Invoice Total	
	Repeat above 3 processes for each item or while OKInvoice is True.	
Purchase Order Total (read from PO file)	Check if Invoice Total matches PO Total	If False then OKInvoice is set to False

Fig 3.18
IPO Chart for the 'Compare with Purchase Order' task in Fig 3.17.



GROUP TASK Describe
Describe in words what happens in the Compare with Purchase Order task modelled in the above IPO diagram.



GROUP TASK Discussion
The IPO diagrams in both *Fig 3.16* and *3.18* describe tasks (subroutines) from the structure chart in *Fig 3.17*. Create IPO diagrams for at least two of the other subroutines on the structure chart in *Fig 3.17*.



GROUP TASK Discussion
Discuss the link between structure diagrams and IPO charts. Use the above IPO chart as an example during your discussion.

DATA DICTIONARY

When developing a software solution to a problem, it is vital that all variable names or identifiers are carefully documented. This ensures that all members of the development team are aware of identifiers that have been used, their data type, storage size, display size and format, scope of usage and various other details. A data dictionary is the repository where this information is held. Data dictionary functions are available with most software development CASE tools. If the data dictionary is an integral part of the development environment then confusion due to duplicate identifiers can be eliminated. For smaller development projects, the data dictionary is often maintained on paper with the assistance of a database or word processor.



Scope
The extent to which a variable is available for use. Global variables are available for use by all subroutines in a program. Local variables are available only within the individual subroutine.

A data dictionary should include a thorough description of every variable and data structure used by the program and each field in each database and file used or accessed by the program. A separate data dictionary is commonly created for each module, database and file used by the program. Often a data dictionary will be a table containing columns for:

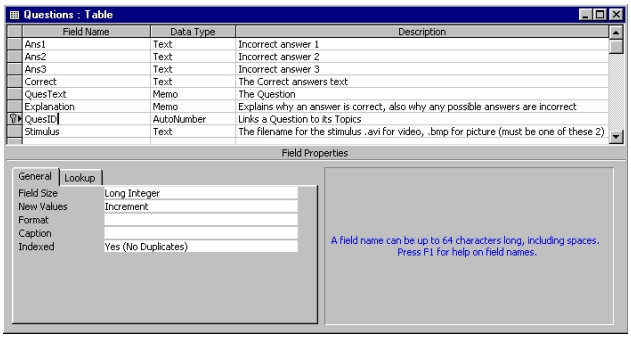


Fig 3.19
A data dictionary from MS-Access

- Identifier name or data item. This is the name used to access the variable within the source code.
- Data type, such as integer, string floating point or Boolean. For data structures the general nature of the structure is used, for example Array(string) specifies an array which contains a sequence of strings. When specifying records use a heading row followed by a separate row for each field within the record.
- Format. Refers to the display format where this is applicable, for example XXNNN for a string indicates a total of 5 characters where the first two characters are alphanumerics and the last three characters must be digits, such as BF345.
- Number of bytes required for storage. For strings each character will use a single byte is ASCII is used and two bytes if Unicode is used. For example, a string containing 5 characters will require either 5 or 10 bytes of storage.
- Size for display. The number of characters required to display the contents of the variable on a screen or on paper. This value will include decimal points, dollar signs, percentage signs or any appropriate characters added to the raw data when it is displayed. For instance, dollar and cents values less than 10,000 will require up to 8 characters to display as $9999.99 contains 8 characters. However, if the data type is single precision floating point then just 4 bytes are required for storage.
- Description or purpose. A brief outline of the logical role the variable plays within the subroutine. May include other relevant details such as whether it is a parameter, input by the user or stored in a file.
- Example. A typical example of an actual data item usually in its display format. This is particularly important for dates, currency and other formats that can be shown in a variety of different ways.
- Validation. Refers to the range of allowable values. For instance, a person's age in years must be positive and less than say 130. Dates are often restricted based on the current date, that is often they must be less than or equal to today's date or for delivery dates they must be after today's date.
- Scope of the variable. Is the variable local to its subroutine or is it available more widely?

The exact columns used will be determined by the nature of the data and how it is used in the source code. A variable that is only used internally (never displayed to users) has no need for a format and size for display, hence these can be left blank. The example data dictionary above in *Fig 3.19* displays a sub-form (with the tabs general and lookup) which allows detailed entry of information specific to the particular data type of the current field. This data dictionary describes a table in a relational database, therefore it includes an icon of a key to indicate the primary key field in the table.

The screen shown in *Fig 3.20* is part of a data flow diagram CASE tool. This screen allows the modification of the attributes of each data item used in the data flow diagram. As data flows are added to the diagrams the data items are automatically added to the data dictionary. Clicking on the 'Edit Data Item' button opens a screen allowing entry of details describing specific data items, such as data type, range, and a description of the data item's purpose.

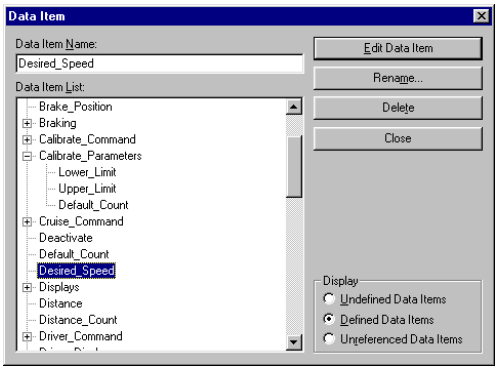
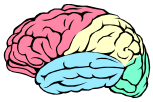


Fig 3.20
Data dictionary screen from the Axiomsys CASE tool.



Consider the following:

The data dictionary below describes all the identifiers used in the Convert Amount to Words subroutine from the Invoicing System structure diagram shown in *Fig 3.17*.

Identifier (Data Item)	Data Type	Number of storage bytes	Description	Example
AmountInWords	String	255	Returns the currency amount in words.	Six thousand, two hundred and fifty dollars and 45 cents.
Amount	Floating point	4	Input parameter.	6250.45
TempDigit	Integer	2	Stores each digit as it is extracted from Amount.	6
DigitWord(19)	Array(string)	10 * 20	The word associated with each digit.	DigitWord(5)="five".
TenMultipleWord(9)	Array(string)	10 * 10	Word for each multiple of ten.	TenMultipleWord(3)="thirty"
Ten3Word(4)	Array(string)	10 * 5	Word for each 3rd power of ten.	Ten3Word(2)="million"
PlaceCounter	Integer	2	Counter incremented for each digit in Amount.	1
TempResult	String	255	Stores the amount in words during processing.	and 45 cents.

Fig 3.21
Data dictionary for the 'Convert Amount to Words' subroutine.

The programmer created this data dictionary whilst developing the source code. For this particular subroutine columns for format and size for display have not been included as the only output is the final amount in words, which is a string up to 255 characters long. Data dictionaries are stored, along with other documentation, to assist in future maintenance and upgrading of the software product.



GROUP TASK Discussion
How could a data dictionary, such as the one above, be of assistance to future maintenance personnel?



GROUP TASK Discussion
Explain the relationship between structure charts, IPO charts and data dictionaries. Use the above data dictionary as an example to illustrate your answer.

SCREEN DESIGNS AND CONCEPT PROTOTYPES

Screen designs provide the interface between the user and the software. Considerations in regard to placement of items, consistency of design, order, grouping and nature of screen elements, use of colour, graphics and fonts, and language used all contribute to the effectiveness of a screen's design. Concept prototypes (see chapter 2) are used to implement screen designs so potential clients can evaluate them. Limited or concept prototypes are designed to gauge the effectiveness or ineffectiveness of screen designs. A concept prototype can be thought of as a working screen design.

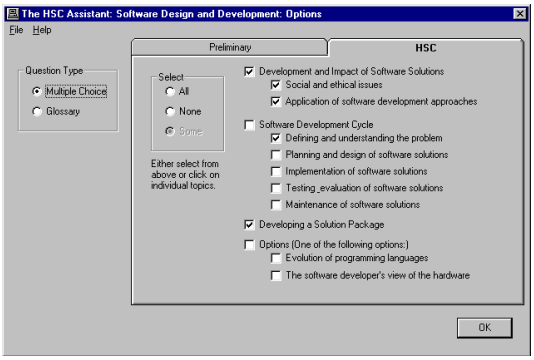


Fig 3.22
Option screen from 'The HSC Assistant: SDD' multimedia CD-ROM for Windows.

Companies that produce operating systems publish standards for screen design to ensure consistency of design between applications. The screen in *Fig 3.22* utilises a number of features complying with Microsoft's standards for Windows screen design. Radio buttons are round, check boxes are square and contain a tick when selected, frames are used to link related items, the screen's title appears in the bar at the top of the screen, a cross is used to close the screen, etc. Adhering to these screen design standards results in reduced learning times for users. Today, many screen elements are similar in function and design across the range of GUI operating systems, however their colour, use of fonts, and format can make them look very different. Compare the above Windows screen with the Apple screen in *Fig 3.23* and the Linux screen in *Fig 3.24*. Many of the elements are similar, however the overall effect is different.

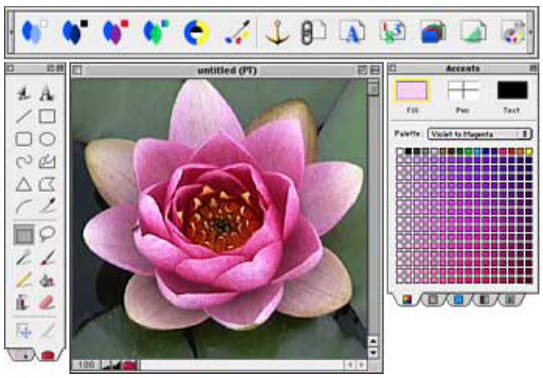


Fig 3.23
Screen shot from the paint module in Appleworks 6 for the Macintosh.

Many integrated CASE tools include areas where preferences in regard to screen design are stored. When source code is generated, the screens will adhere to the preferences entered into the CASE tool. This encourages consistency of design across all screens in a particular application.

Many large-scale business applications use a text-based user interface. Text-based screens are often designed using a paper grid containing a cell for each character that makes up the screen.

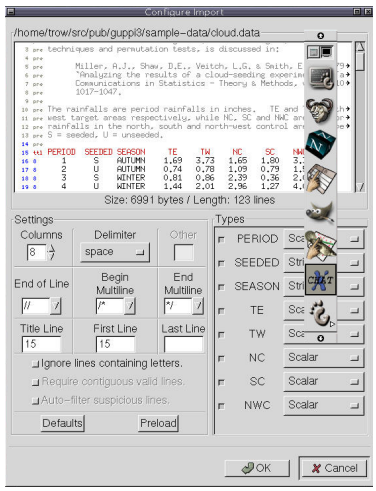


Fig 3.24
Screen from an application for the Linux operating system.



GROUP TASK Activity
Use the internet to find information about standards for screen design for the operating system you most commonly use. Compile a list of your findings.

STORYBOARDS

A storyboard is used to describe the screens used in a project together with how each screen interacts with other screens. Effectively, a storyboard for a software product is a collection of screen designs together with a diagram describing the links between the screens. Storyboards are commonly used in movie production where a series of cartoon-like screens are produced. For movie production, the sequence in which screens are encountered is always linear, as the movie needs to be watched from start to finish to make sense. This is not the case with most software. The interactive nature of software means that storyboards must carefully document the possible flow of data between screens.



Fig 3.25
A storyboard for a movie is always linear.

Storyboards for software products usually include a series of screen designs together with a diagram illustrating the possible navigation paths between screens. This diagram will show each screen as a rectangle with arrows leading to and from other screens. If the screens are small and simple and the screen designs fit on a single page then the links can be included without the need for a separate diagram. It is important to develop a logical navigation system between screens so users do not become lost in a maze of screens. Navigation buttons are normally included on every screen in a consistent position to assist users.

There are many ways of organising the navigation between screens when developing a new application. The most common method is to utilise a hierarchical structure where a main menu directs the user to the major screens of the application. Each of these screens will have further links to lower-level screens in the hierarchy.



Consider the following:

The diagram below shows a hierarchical screen structure for a company's web site. Although the structure is basically hierarchical, a link is provided on each screen to return the user to the home page.

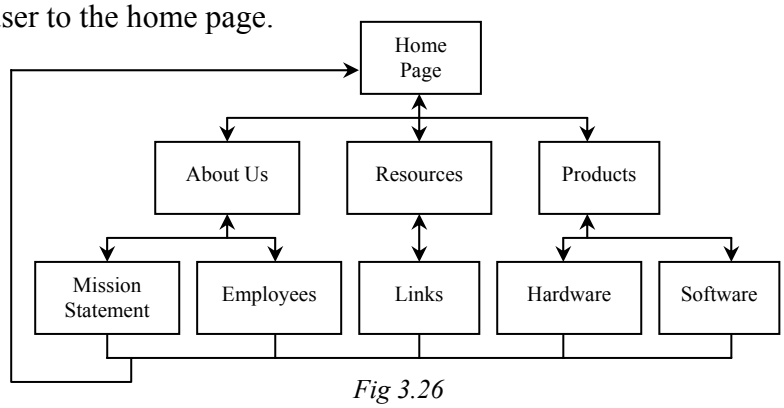


Fig 3.26
Hierarchical navigation between screens is common for web sites.



GROUP TASK Discussion
Is there any way of viewing the Hardware page without first viewing the Products page? Suggest reasons why the developer may have done this.



Consider the following:

The storyboard below describes the navigation between the screens of a school's reports package, together with three screen designs that formed part of the original concept prototype for the project.

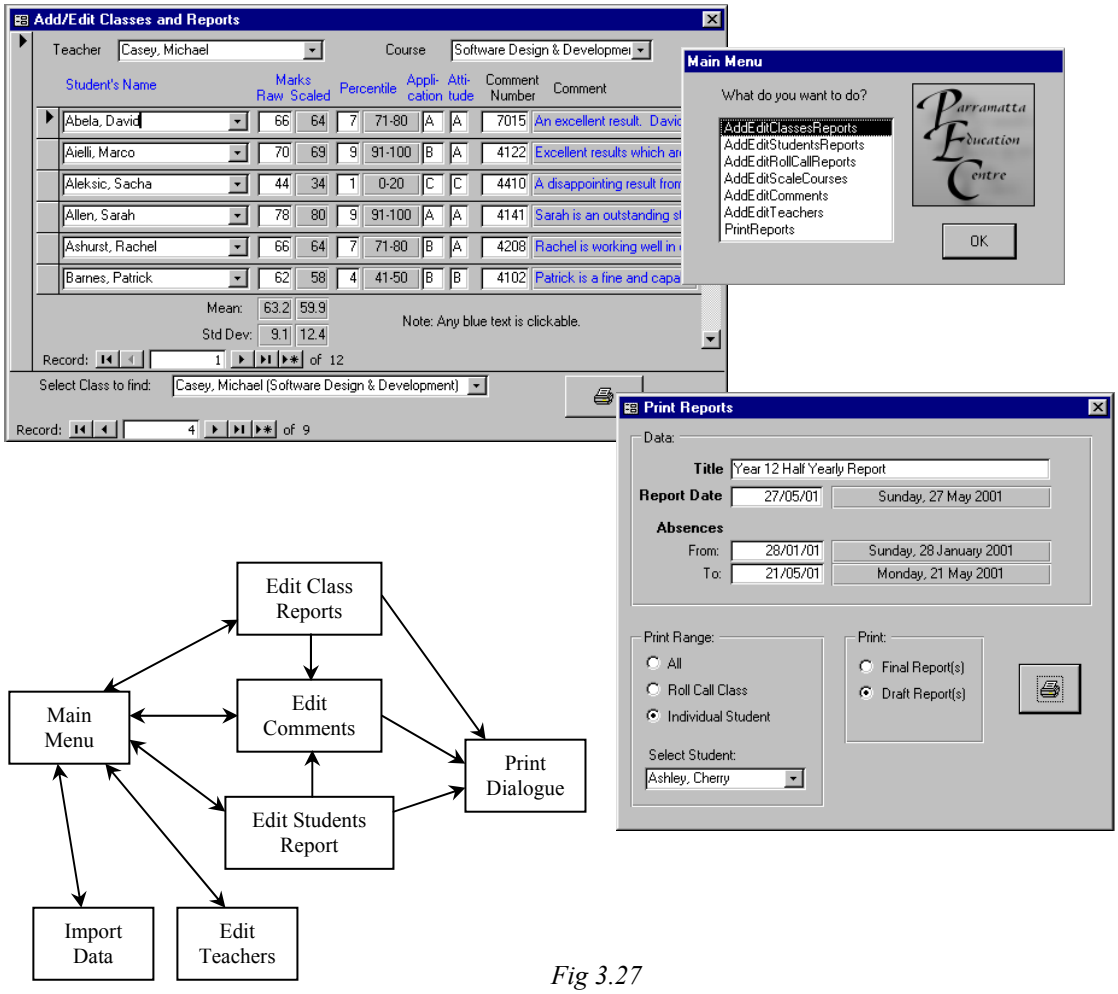


Fig 3.27
Storyboard for a school's computerised report system.



GROUP TASK Discussion
List and describe the screen elements shown on the above three screen designs. Do you have any suggestions for improving their design?



GROUP TASK Activity
Initial screen designs are often done on paper. Create a new design for the Main Menu using your knowledge of sound screen design.



HSC style question:

A graphics conversion program is being developed to convert images between the BMP, JPG and GIF formats. The program will convert all graphic files of a particular format that are found within a specified directory into one of the other graphic formats. For example, if C:\Images is the chosen directory and the user wishes to convert from BMP to JPG then all BMP images in the C:\Images directory are converted into JPG images.

(a) Create an IPO diagram to describe the overall operation of this graphics conversion program.

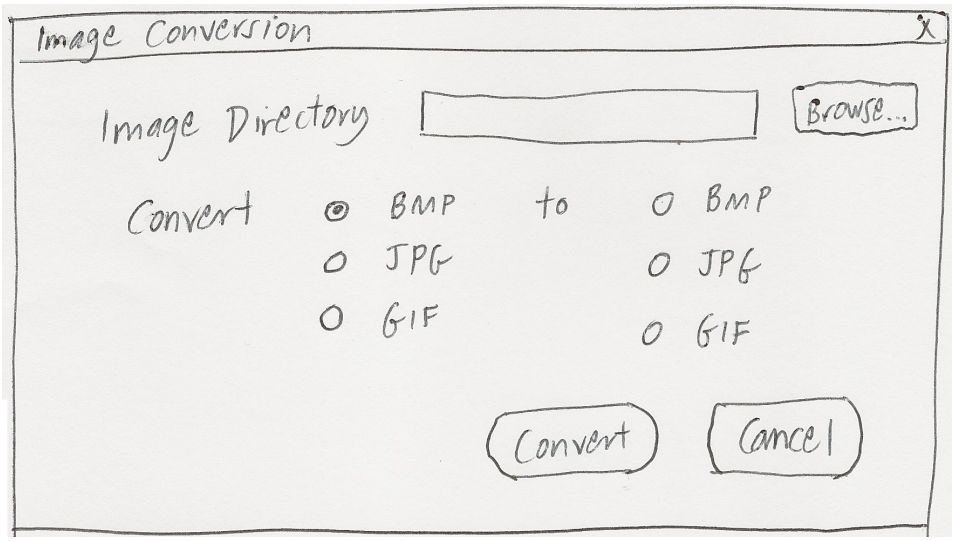
(b) Create a screen design for the graphics conversion program.

Suggested solutions

(a)

Input	Process	Output
Directory name File type old File type new	• Get directory name • Get input (old) and output (new) file types (JPG, BMP or GIF) • Determine and store file names found in the directory that are of the relevant input file type • Convert each image in turn into the desired output file type, saving using the same name but with the new file extension	Converted graphic files in new file type

(b)



SET 3C

1. A model of a system can best be described as:
   (A) a system flowchart representing logic and movement of the data.
   (B) a representation of a system showing structure and functionality.
   (C) a data dictionary describing the data types used by a system.
   (D) a structure chart showing the navigation between screens.

2. IPO diagrams explain:
   (A) the data types used.
   (B) the screen designs.
   (C) how inputs are transformed into outputs.
   (D) the flow of data.

3. Data flow diagrams explain:
   (A) the data path through the system.
   (B) the timing of the data path through the system.
   (C) the manual processes component of the system.
   (D) the sequence of processing.

4. Structure charts are used to:
   (A) model the hierarchy of the processes within the system.
   (B) show the algorithms that the system will use.
   (C) determine user requirements.
   (D) all of the above.

5. A data dictionary should include:
   (A) identifiers.
   (B) data types.
   (C) number of bytes of storage.
   (D) all of the above.

6. The interface between the end-user and the software is largely determined by:
   (A) the operating system.
   (B) the screen designs.
   (C) the clients evaluating the system.
   (D) the source code.

7. Concept prototypes are useful because they:
   (A) result in reduced learning times for users.
   (B) use all the same elements.
   (C) assist in gauging the effectiveness of the screen designs.
   (D) are all text based screens.

8. A collection of screen designs together with a diagram describing the links between screens is referred to as a(n):
   (A) storyboard.
   (B) IPO diagram.
   (C) structure chart.
   (D) hierarchical diagram.

9. Data dictionaries and other documentation should be stored because:
   (A) it assists in the maintenance of the software.
   (B) it assists in the upgrading of the software.
   (C) it assists in understanding the processes used by the software.
   (D) all of the above.

10. Which representation is generally used at a higher level giving an overall view of the system?
    (A) system flowcharts.
    (B) storyboards.
    (C) data flow diagrams.
    (D) IPO diagrams.

11. Identify each of the symbols used in system flowcharts. Give an example of a situation where each symbol may be used.

12. Draw a system flowchart to model a system of your choice. Use at least 6 of the symbols identified in question 11.

13. Structure charts have many different names. Compile a list of these and explain in your own words how each of these terms describes the nature of a structure chart.

14. Traditional board or card games have very definite rules. For this reason they are good examples to use when learning to apply the logic and structure that is involved in software development. Using this concept, draw a structure diagram for a board or card game with which you are familiar.

15. Data flow diagrams are particularly useful for describing systems that are data oriented. An EFTPOS (Electronic Funds Transfer Point Of Sale) terminal is used to process withdrawals directly from customers' accounts. These terminals are now commonplace in most retail stores.

    Design a data flow diagram to model the operations of a typical EFTPOS terminal.

QUALITY ASSURANCE

Quality assurance (QA) is a set of processes that occur throughout the development and manufacturing of any type of product. These QA processes ensure standards and requirements are met and continue to be met such that the final products are of a high quality. For most manufactured products quality assurance is largely about monitoring the manufacturing process to ensure parts continue to meet tolerances and standards over time. Random samples can be removed from the production line for thorough testing. Software is somewhat different as it is the design and development process that determines quality. Software quality is a measure of the extent to which desirable features, including functional requirements, are included such that the software reliably and efficiently continues to meet the needs of users. Software Quality Assurance (SQA) aims to provide users with confidence in the integrity of software. SQA ensures that software will function reliably as intended and is free of errors and other vulnerabilities. Vulnerabilities include both intentional and unintentional faults, such as security flaws, inclusion of untested code and of course a variety of other bugs which may result in loss of data or reduced reliability.



Software Quality
The extent to which desirable features are incorporated in to software so it can reliably and efficiently continue to meet the needs of users.

How can software quality be judged throughout the design and development of software? A set or list of measurable criteria is required. The criteria will be different and have different focus depending on the nature and size of the software project. Software which will operate critical systems, such as the electricity grid or an aircraft's flight controls, requires a particular focus on reliability and accuracy whereas a graphics editor may focus more on usability. The following list includes a range of areas that should be assessed or at least considered when developing quality assurance criteria.

- Efficiency. The software should perform its functions by making the best use of the computer's resources. Judging efficiency includes assurance that good programming techniques are used such as the use of algorithms which optimise processing time, well structured code and thorough testing. In addition the selection of an appropriate language for the problem and the use of operating system and CPU features such as multi-tasking, distributed and parallel processing. Efficiency should be judged using a range of possible system configurations, such as different versions of operating systems, amount of RAM, video RAM and various CPUs. Code that accesses networks, particularly the internet will often greatly affect efficiency and should be thoroughly examined and optimised.
- Integrity. The correctness of data within the system. Integrity is improved when data validation is thorough and the ability to access the system illegally or inappropriately is tightly controlled. Data integrity will often reduce over time as time dependant data becomes out of date. Systems to check such data should be included, for instance checks that phone numbers, addresses and email addresses are current. During development integrity can be judged based on the security mechanisms for accessing the system and also on data validation as data is entered into the system.

- Reliability. The ability of the system to continue to perform its functions over time. Other reliability factors that should be considered include the time taken to recover from a failure and the ability of the system to be updated to reflect changes in requirements or changes in the environment in which the software operates. For instance, a new version of an operating system may cause issues for the software or a hard disk may fail. Such problems should be able to be fixed in a timely manner if the software is to perform reliably.
- Usability. The ability of software to be learnt and used by users. The design of the user interface is critical when judging usability. Factors such as the consistency of the design and the correct use of screen elements and colour. Other considerations include recovery from errors, warnings or undo features when data changes of significance are made and also features for advanced users such as shortcut keys and the ability to record macros which can be replayed to repeat commonly performed tasks. For instance a macro might generate a weekly report, print it to a pdf file and email it to the appropriate recipients.
- Accuracy. The software performs its functions correctly and according to its specifications. Accuracy can only be assured when the code has been thoroughly tested. Algorithms can be assessed to ensure correctness for all possible paths and sets of inputs. The code should be well commented so that accuracy can be ensured during future upgrades.
- Maintainability. Most successful software will undergo changes throughout its lifetime. Maintainability is a measure of the ease with which these changes can be made. Source code should be well documented, modular in design and test data or test routines should be retained so thorough testing can be performed after any code changes have been made. There are three main reasons code is routinely altered – to correct errors, to implement new functionality or to improve performance. Code should always be written with future maintenance in mind. Often the logic within code appears obvious when first written but is difficult to comprehend later when modifications are required.
- Testability. The ability to test all aspects of the software is critical to each of the other quality assurance factors. Testing occurs at various times and levels as software development progresses. Individual subroutines and then modules should be thoroughly tested, the system should be tested to ensure it performs according to requirements and finally the client performs acceptance testing to ensure requirements have been met with a live system.
- Re-usability. The ability to reuse code in other related systems. Once a fully tested routine or module has been produced it can be reused as part of future software projects. Often minimal effort is required to write a more general case of a required subroutine which is much more suited to reuse. Most programmers maintain a library of subroutines and modules that perform commonly used functions. There is no need to reinvent the wheel and furthermore the accuracy of future software is improved when thoroughly tested routines are reused. There are many commercial and free libraries of code available for most languages. Portability is another significant aspect of reusability. Many languages are compiled so the same code can be used for a variety of operating systems.



GROUP TASK Discussion
Does each of the above areas improve software quality for software developers or for their clients? Discuss.

Quality and often people's perception of quality has an enormous impact on the success of all products including software. It is critical that management processes are put in place to ensure quality criteria is met throughout the software development cycle. Quality assurance is an ongoing process throughout design and development which ensures the identified quality criteria are met.



Consider the following new software products:

- Software driver for a new web camera.
- Record system for breeders of exotic animals.
- Software for a new hand held GPS.
- New version of Microsoft Word.



GROUP TASK Discussion
For each of the above software products identify and justify quality assurance criteria likely to be of particular importance.



GROUP TASK Discussion
For each of the quality criteria identified above, discuss suitable management processes that could be used to ensure the quality criteria is met during the software development cycle.



HSC style question:

(a) Define the term "Quality Assurance" as it applies to software development.

(b) Maintainability is one aspect of quality software. Explain how maintainability can be assured during the development of software.

Suggested solutions

(a) Quality assurance is an ongoing process occurring throughout the software development cycle to ensure software is of high quality. Quality criteria are set and are then monitored during development. The quality of the product is judged according to the extent to which the quality criteria have been met.

(b) Maintainability is the ease by which a software product can be altered to correct errors, improve performance or add new functions. To assure maintainability the source code should be well documented with comments and the use of intrinsic or meaningful variable names. In addition system models such as structure charts and DFDs should be kept and updated. Also a strict system of testing should be implemented and these tests kept so they can be rerun when any changes are made to the code. All these aspects of maintainability should be specified as requirements to all developers. They should be checked and monitored on an ongoing basis during the development process so that maintainability is assured.

CHAPTER 3 REVIEW

1. The first stage of the software development cycle is:
   (A) choosing a method that is appropriate to the construction of the solution.
   (B) defining and understanding the problem.
   (C) selecting the most suitable personnel for the project.
   (D) developing the design specifications for the system.

2. Items that are not part of a system are said to be:
   (A) part of the environment.
   (B) part of the boundaries of the system.
   (C) part of the interface.
   (D) part of the limits of the system.

3. The difference between boundaries and interfaces is:
   (A) boundaries are factors with which the system must comply; interfaces are the limits of the system.
   (B) boundaries are what you are aiming to achieve; interfaces are what is keeping you from achieving them.
   (C) boundaries are the limits of the system; interfaces are how the system communicates across a boundary with its environment.
   (D) boundaries are within the system; interfaces are within the environment.

4. A software is difficult to maintain and has usability issues. Which of the following would have helped minimise these issues during the initial development processes?
   (A) thorough system testing.
   (B) quality assurance.
   (C) careful problem definition.
   (D) needs analysis.

5. Social and ethical issues requiring consideration as part of a software development project include:
   (A) legal issues.
   (B) effects on employment levels.
   (C) the changing nature of work.
   (D) all of the above.

6. Joe and Mary meet after work for dinner and discuss their fears regarding the new system that is being implemented at their place of work. Mary believes that she will be made redundant when the new system is operational. Her fears are commonplace in the given scenario and fall under the category of:
   (A) social and ethical issues.
   (B) legal issues.
   (C) technical issues.
   (D) operational issues.

7. The diagram that describes the top-down design and sequence of processing is called a(n):
   (A) IPO diagram.
   (B) storyboard.
   (C) structure chart.
   (D) data flow diagram.

8. Processes can best be described as:
   (A) the destination of data.
   (B) a model of a system.
   (C) the interface between the user and the system.
   (D) the actions that take place on the data within the system to transform inputs into outputs.

9. A need is often expressed in emotive, non-specific terms. Which need listed is a clear example of this?
   (A) Our image in the marketplace needs improvement.
   (B) Our profit margin needs to increase by 15% over 18 months.
   (C) The new system needs to be implemented by July 15.
   (D) Our call out service fee should increase by $5.00.

10. A requirement should be able to be readily tested to determine whether it has been achieved. Which requirement cannot be readily tested?
    (A) 95% of deliveries are shipped within 3 working days.
    (B) Clients are contacted at least once every 6 months.
    (C) Customer perceptions of our company are vastly improved.
    (D) Data entry productivity is increased by 15%.

11. A software company is examining the possibility of developing a product that will enable high-speed internet access in hotel rooms. The software product will provide high-speed networking capabilities as well as monitoring each guest's online time. The online time will be sent to the hotel's main computer where charges will be debited to the guest's account. At the present time, hardware is available to complete this task. A cable network already exists into each room to deliver cable television. The new product would involve a dedicated internet connection being installed in each of the hotel's rooms.

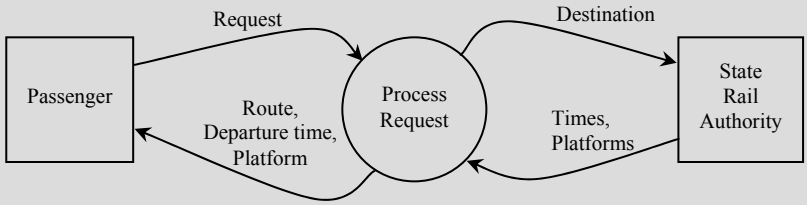


    Describe at least four aspects of this proposal that should be closely examined as part of defining and understanding the problem.

12. Design specifications are developed to ensure the development process adheres to certain standards. These specifications determine a framework in which members of the development team will operate.

    List and describe at least 6 items that should be considered as part of the design specifications for a software development project.

13. The SRA (State Rail Authority) is looking to develop a computerised timetable system that will allow passengers to enter their destination and obtain details of the route, time of departure and platform of the next available train. A context diagram is provided below. Use this context diagram as an aid in the creation of a level one data flow diagram for this scenario.



14. As part of the development team for a new software product, you have been given the following IPO diagram. Use this IPO diagram to develop an algorithm that performs this task.

Client: Build Soft Pty. Ltd.
Project: Encryption System
Module: FindTwoPrimes

IPO DIAGRAM

Date: 18/10/2001
Programmer: Youcan Doit
Page: 3 of 9

Input	Process	Output
Lowest as an integer, the two primes must be greater than this value	Loop around for each integer greater than Lowest.	
	For each integer, loop around from 1 to the square root of the integer.	
	Check if any of these numbers divide evenly into the number being checked.	
	If no numbers go in evenly then the integer is a prime so keep it.	
	Do the same to find a second prime.	Prime1, Prime2

Notes: A prime number only has factors of 1 and itself. E.g. 1 and 7 are the only integers that go into 7 so 7 is a prime number. 1, 2, 3 and 6 go into 6 so 6 is not a prime number.

15. In question 13, we created a data flow diagram for a computerised timetable system for the State Rail Authority. Assuming that this system will be implemented using a touch screen, design a series of screens for this product. Remember, people with various forms of disability will use this system. Justify your choice of screen design elements.

In this chapter you will learn to:

- recognise the logic in a standard approach (such as a sort or search)
- apply standard approaches as part of the solution to complex problems
- read, interpret and modify algorithms developed by others
- document the logic required to solve problems, including:
  - nesting of control structures
  - record structure
  - the use of files (sequential and relative)
  - random number generators
  - arrays of records
  - multidimensional arrays
- develop a suitable set of test data
- desk check algorithms and source code that include complex logic
- select an appropriate data structure to solve a given problem
- develop and appropriately document a module for use by others
- correctly incorporate a standard module into a more complex solution, passing parameters effectively
- represent a software solution in diagrammatic form
- interpret and modify existing system modeling diagrams
- select and use appropriate software to assist in the documentation of a software solution
- recognise the relevance of CASE tools in the planning and design of a software solution
- design and evaluate effective interfaces for software solutions
- use a RAD environment to produce user interfaces
- recognise that the choice of programming language to be used depends on the problem to be solved
- interpret a benchmark report to select the most suitable technology for a specified task
- produce a benchmark report for a simple iterative process running under two different environments or conditions

Which will make you more able to:

- explain the interrelationship between hardware and software
- describe how the major components of a computer system store and manipulate data
- identify and evaluate legal, social and ethical issues in a number of contexts
- construct software solutions that address legal, social and ethical issues
- identify needs for which software solutions are appropriate
- apply appropriate development methods to solve software problems
- apply a modular approach to implement well-structured software solutions and evaluate their effectiveness
- apply project management techniques to maximise the productivity of the software development
- create and justify the need for the various types of documentation required for a software solution
- select and apply appropriate software to facilitate the design and development of software solutions
- communicate the processes involved in a software solution to an inexperienced user
- use and describe a collaborative approach during the software development cycle.
- develop and evaluate effective user interfaces, in consultation with appropriate people.

In this chapter you will learn about:

Standard algorithms for searching and sorting

- standard logic used in software solutions, namely:
  - finding maximum and minimum values in arrays
  - processing strings (extracting, inserting, deleting)
  - generating a set of unique random numbers
  - processing of sequential files, including sentinel value, priming read, open for input, output or append, close, appending records
  - processing of relative files, including open for relative access, defining a key field for a relative file, retrieving, writing and updating a record in a relative file
  - linear search, binary search
  - bubble sort, insertion sort, selection sort

Custom-designed logic used in software solutions

- requirements to generate these include:
  - identification of inputs, processes and outputs
  - representation as an algorithm
  - testing of the logic in the algorithm
  - identification and definition of required data structures
  - use of data structures, including multidimensional arrays, arrays of records, files (sequential and relative)
- customised off-the-shelf packages
  - identifying an appropriate package
  - identifying the changes that need to be made
  - identifying how the changes are to be made

Standard modules (library routines) used in software solutions

- reasons for the development and use of standard modules
- requirements for generating a module or subroutine for re-use, including:
  - identification of appropriate modules or subroutine
  - appropriate testing using drivers
  - thorough documentation of the routine: author, date, purpose
  - order and nature of parameters to be passed
- issues associated with reusable modules or subroutines, including:
  - identifying appropriate modules or subroutines
  - considering local and global variables
  - appropriately using parameters (arguments)

Documentation of the overall software solution

- tools for representing a complex software solution including:
  - algorithms
  - refined system modeling tools, including IPO diagrams, context diagrams, data flow diagrams (DFDs), storyboards, structure charts, system flowcharts, data dictionaries

Interface design in software solutions

- the design of individual screens in consultation with the client, including:
  - consideration of the intended audience
  - identification of screen size
  - identification of data fields and screen elements required and their appropriate on-screen placement
  - online help
  - consistency in approach
  - recognition of relevant social and ethical issues
  - current common practice in interface design. Radio buttons, check boxes, dropdown lists scroll bars, mobile phone interface elements

Factors to be considered when selecting the programming language to be used

- sequential or event-driven software.
  - driven by the programmer or user
- features required, and features available in the language
- commands within the language to interface with the required hardware
- ability to run under different operating systems

Factors to be considered when selecting the technology to be used

- performance requirements
- benchmarking

4

PLANNING AND DESIGNING SOFTWARE SOLUTIONS

Planning and designing software solutions is the second stage of the software development cycle. This is the stage where data structures are designed and algorithms for the final solution are created. Often modules or subroutines from previous solutions or from libraries of commonly used modules will be incorporated into the solution.

Models of the system developed during the first stage of the software development cycle are refined to reflect the detail of the system's design. Individual team members will be allocated particular modules and subroutines to complete. Data flow diagrams, structure diagrams, IPO charts and screen designs provide the information necessary for team members to develop the data structures and logic necessary to fully develop each subroutine.

In this chapter, we examine a number of standard algorithms, introduce some new data structures, examine file processing and then consider some problems requiring custom-designed logic. We then consider the design of the user interface, followed by factors to consider when selecting a suitable programming language. Finally we consider benchmarking as a technique for assessing different technologies.

Before we continue let us review some important concepts from the preliminary course.

What is an algorithm?

An algorithm is a method of solution for a problem. Algorithms describe the steps taken to transform inputs into the required outputs. Each algorithm will have a distinct start and end, and will be composed of the three control structures: sequence, decision and repetition. Algorithms are described using an algorithm description language; in this course we use either pseudocode or flowcharts. Pseudocode and flowcharts provide a language in which we can express algorithms.

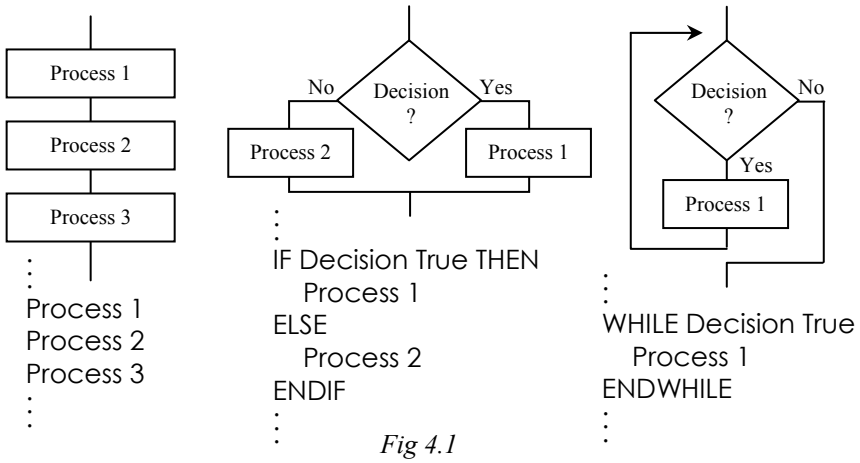


Fig 4.1
The three control structures: sequence, decision and repetition.

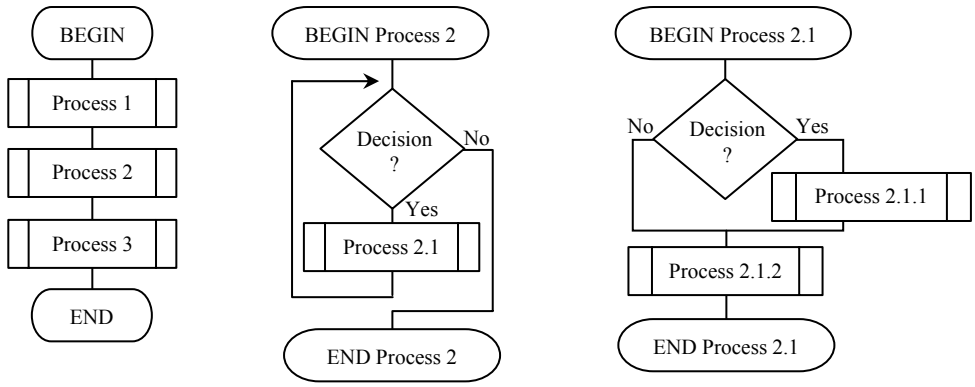


GROUP TASK Activity
In *Fig 4.1*, multiple selection and post-test repetition are not shown. Create flowchart and pseudocode examples for each of these control structures.

What is top-down design?

Top-down design is the process of breaking down a problem into its component parts. Each component part or module is further broken down until a level is reached where the parts can be implemented in a programming language. This process is known as stepwise refinement: each level or step is a refinement of the previous level. Structure charts provide a modelling technique for describing the top-down design of a solution.

An algorithm is written for each component or subroutine of the top-down design. The syntax used in pseudocode and flowcharts is illustrated below.




```
BEGIN                    BEGIN Process 2          BEGIN Process 2.1
   Process 1                WHILE Decision True      IF Decision True THEN
   Process 2                   Process 2.1              Process 2.1.1
   Process 3                ENDWHILE                 ENDIF
END                         END Process 2            Process 2.1.2
                                                     END Process 2.1
```

Fig 4.2
The top-down design of a solution expressed using flowcharts and pseudocode.

What are functions and procedures?

The algorithm for each subroutine of the solution will be implemented in a programming language as either a function or a procedure. Functions and procedures are called by higher level modules. Procedures return their results via parameters. Functions return one result via the function itself. In essence, one can think of the function's name as a variable with a data type just like any other variable. Functions are used as parts of expressions; procedure calls are statements on their own. There are functional languages, such as Lisp and Haskell which are entirely based on functions. Many languages are object oriented where subroutines are written as methods of classes – methods are similar to functions as they return a result.

For example, if a function called Average is written, which calculates the average of an array of numbers, then Result = Average(Marks) would set the variable Result to the Average of the elements contained in the array called Marks. A procedure written to achieve the same result would require two parameters: one for the input and a second parameter to return the result. Average(Marks, Result) would return the average via the second parameter, and the variable Result would be set to the average of the elements contained in the array Marks. In this example, a function would be a better choice than a procedure; however both will achieve the same result.

STANDARD ALGORITHMS

Many problems encountered by software developers use similar techniques as part of their solution. It makes sense to develop standard methods of approach rather than continuing to reinvent the wheel. These standard modules can be included in a library of routines for future use. In this section, we examine a number of standard algorithms that perform tasks such as searching and sorting data. We then consider string processing, random numbers and finally processing files. It is more important to understand the concept behind each of these algorithms than to memorise the algorithms themselves.

We will examine standard algorithms for each of the following:

- Linear search
- Finding maximum and minimum values
- Binary search
- Insertion sort
- Selection sort
- Bubble sort
- Processing strings (extracting, deleting and inserting)
- Generating a set of unique random numbers
- Processing of sequential and relative files



Consider Big O notation:

Big O notation is used to compare algorithms. It describes the performance or time complexity of algorithms compared to the number of data items being processed. The mathematical formulation of Big O for individual algorithms is beyond the SDD course; however a general understanding of the most common classifications will assist when comparing the performance of standard and custom algorithms. Big O describes the worst case behaviour of an algorithm, such as searching for an item that isn't present in the data.

Common Big O classifications from fastest to slowest are briefly described below followed by a graph and table to illustrate the relative differences.

- O(1) – The processing time is constant regardless of the number of data items. For example, displaying the first data item in an array is O(1). It takes the same time regardless of the size of the array.
- O(log *n*) – Processing time increases slower as *n* increases in size. For instance, it may take 1ms (millisecond) to process 10 data items, 2ms to process 100 items and 3ms to process 1000 data items. The binary search is an example of an O(log *n*) algorithm.
- O(*n*) – Processing time is proportional to the number of data items. For example, if the number of data items is doubled then the time also doubles. This is very common as any algorithm which performs processing on each data item will be O(*n*). Linear searches are an example of an O(*n*) algorithm.
- O(*n* log *n*) – Processing of each data item is an O(log *n*) process. As there are n data items then the time complexity is O(*n* log *n*). Efficient sorting algorithms are O(*n* log *n*), for example Quick sort and Merge sort.

- O(n^2) – Processing of each data item potentially requires examining all other data items. This means time increases rapidly as the number of data items *n* increases. The three sorts we examine in SDD are all O(n^2).
- O(2^n) – Processing time increases exponentially as the number of data items *n* increases. O(2^n) algorithms are of little use unless you are certain *n* will always remain small.

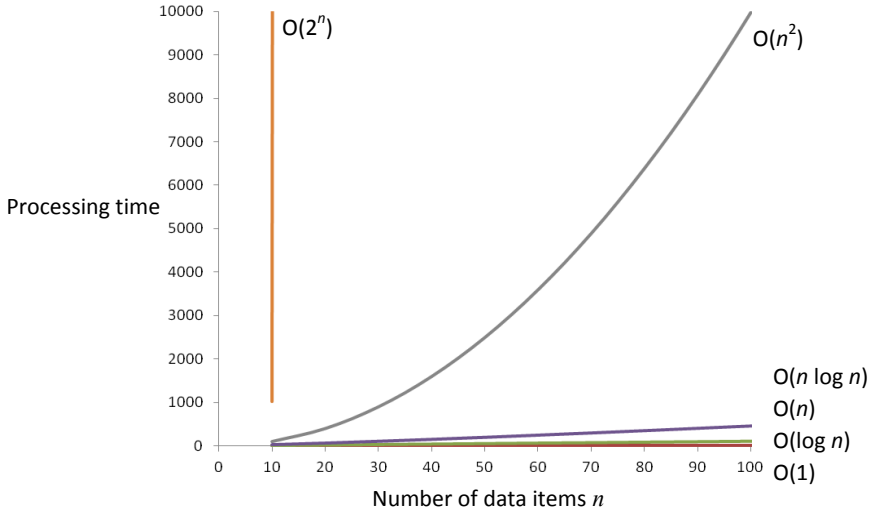


Fig 4.3
Graph showing the relative differences for selected Big O algorithms.

The graph in *Fig 4.3* shows the scale of the differences between each classification and these enormous differences become increasingly more significant as *n* increases. The graph shows relative processing time for up to just 100 data items. Many algorithms used on a daily basis process many millions of data items, such as when processing graphics, sound and video files. For millions of data items the difference in time between O(log *n*), O(*n*) and O(*n* log *n*) becomes highly significant. The table in *Fig 4.4* illustrates the magnitude of the differences in processing time as *n* increases by a factor of ten. For instance, if you can reduce an O(*n*) algorithm to an O(log *n*) algorithm then when processing 1 million data items you reduce the time taken from 1 million units of time down to just 14 units of processing time - a staggering saving. If the algorithm is processing pixels within a bitmap image on a mobile phone then potentially a few seconds of processing is reduced to a virtual instant at no cost.

Data items	Processing time				
n	O(1)	O(log *n*)	O(*n*)	O(*n* log *n*)	O(n^2)
10	1	2	10	23	100
100	1	5	100	461	10000
1000	1	7	1000	6908	1000000
10000	1	9	10000	92103	100000000
100000	1	12	100000	1151293	10000000000
1000000	1	14	1000000	13815511	1000000000000
10000000	1	16	10000000	161180957	100000000000000
100000000	1	18	100000000	1842068074	10000000000000000
1000000000	1	21	1000000000	20723265837	1000000000000000000
10000000000	1	23	10000000000	230258509299	10000000000000000000
100000000000	1	25	100000000000	2532843602293	1000000000000000000000

Fig 4.4
Differences in big O processing time as n increases by a factor of 10.

LINEAR SEARCH

The word linear means straight line. A linear search involves examining items one at a time beginning at the start of the list and continuing to the end of the list. Linear searches do not require the data to be in any particular order. A list containing 100,000 items will require between 1 and 100,000 comparisons to find a specific item if it exists within the list. For situations where items are not found or we wish to determine the number of times an item exists then all items in the list must be examined. As a consequence, the time taken for linear searches to complete is directly proportional to the number of elements in the list. For instance, searching a list of 2 million items will take double the time taken to search a list of 1 million items. In Big O notation a linear search is an O(n) algorithm. n is the number of items in the list and O(n) essentially means there is a straight line which converts the number of items n to the processing time required O(n).



The concept

Each item in the array is examined in turn. The word "Found is displayed each time the required item is encountered.

```
BEGIN LinearSearch
    Set Count to 0
    Get ItemToFind
    WHILE more items in Item array
        IF Item(Count) = ItemToFind THEN
            Display "Found"
        ENDIF
        Add 1 to Count
    ENDWHILE
END LinearSearch
```

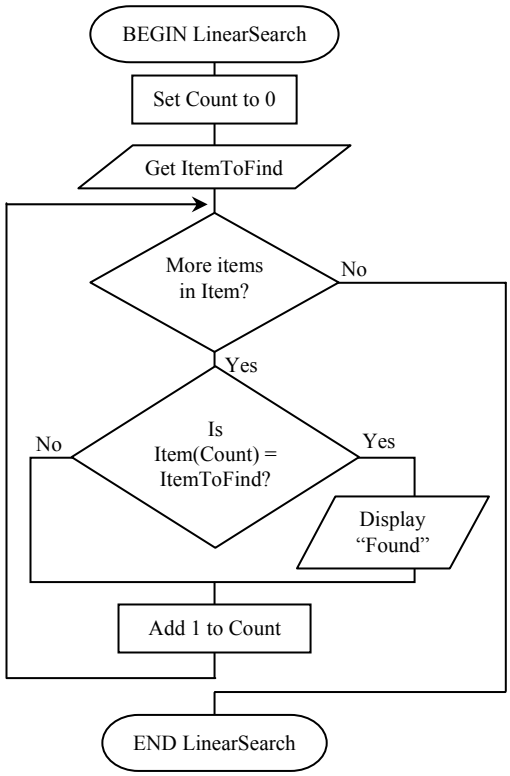
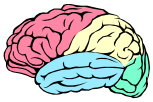


Fig 4.5
Pseudocode and flowchart for a linear search



Consider the following linear search algorithms:

```
BEGIN LinearSearch(ItemToFind, Item)
    Set Count to 0
    Found = False
    WHILE more items in Item AND Not Found
        Found = (Item(Count) = ItemToFind)
        Add 1 to Count
    ENDWHILE
    RETURN Found
END LinearSearch
```

```
BEGIN LinearSearch(ItemToFind, Item)
    Set Count to 0
    NumFound = 0
    WHILE more items in Item
        IF(Item(Count) = ItemToFind) THEN
            Add 1 to NumFound
        ENDIF
        Add 1 to Count
    ENDWHILE
    RETURN NumFound
END LinearSearch
```



GROUP TASK Discussion
Explain how each of the linear search algorithms above differ from the algorithm shown in *Fig 4.5*.

FINDING MAXIMUM AND MINIMUM VALUES

Finding maximum and minimum values in an unsorted list of items is essentially performed using a linear search where a record is maintained of the largest or smallest value as the list is traversed. For sorted lists, finding maximum and minimum values is trivial: they will be either the last or the first values in the list.



The concept

The first item in the array is initially set as the maximum (or minimum) value. Each item in the array is compared in turn with the maximum (or minimum) value. If an item is found to be larger (or smaller) then it becomes the maximum (or minimum) value. This process continues until the end of the list is reached.

```
BEGIN FindMaximum
    Set Count to 1
    Set Maximum to 0
    WHILE more items in Item array
        IF Item(Count) > Item(Maximum) THEN
            Set Maximum to Count
        ENDIF
        Add 1 to Count
    ENDWHILE
    Display Item(Maximum)
END FindMaximum
```

```
BEGIN FindMinimum
    Set Count to 1
    Set Minimum to 0
    WHILE more items in list
        IF Item(Count) < Item(Minimum) THEN
            Set Minimum to Count
        ENDIF
        Add 1 to Count
    ENDWHILE
    Display Item(Minimum)
END FindMinimum
```

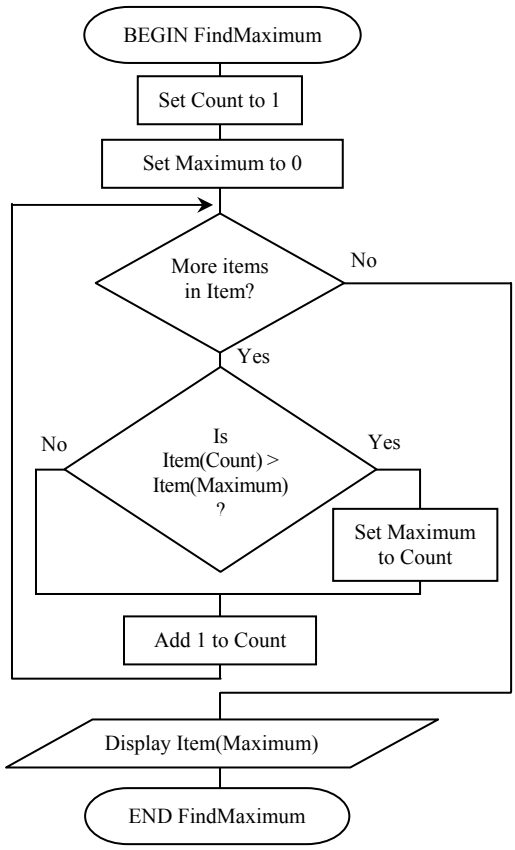


Fig 4.6
Pseudocode for finding maximum and minimum values and flowchart for finding maximum value.



GROUP TASK Activity
In the above algorithms is the Item array indexed from 0 or from 1? Perform a desk check of the algorithms to confirm your answer.



GROUP TASK Activity
Using a flowchart, write an algorithm to find the minimum value by searching from the end of the list to the beginning of the list.



Consider the following:

A list of 30 unsorted integers between 1 and 5 is stored in an array. As part of the solution to a problem it is necessary to search this array and find the number of times each integer occurs in the array.

An algorithm has been written to perform this task and is reproduced below:

```
BEGIN TallyIntegers
    Set Count to 1
    WHILE Count <= 30
        Total(Item(Count)) = Total(Item(Count)) + 1
        Add 1 to Count
    ENDWHILE
END TallyIntegers
```

The following set of data is used to perform a desk check of the above algorithm:

1, 2, 2, 5, 5, 4, 3, 4, 5, 3, 2, 2, 3, 1, 4, 4, 3, 3, 1, 5, 3, 2, 4, 1, 3, 4, 3, 5, 1, 3

Following is the desk check:

Count	Item(Count)	Total(1)	Total(2)	Total(3)	Total(4)	Total(5)
1	1	1				
2	2		1			
3	2		2			
4	5					1
5	5					2
6	4				1	
7	3			1		
8	4				2	
9	5					3
10	3			2		
11	2		3			
12	2		4			
13	3			3		
14	1	2				
15	4				3	
16	4				4	
17	3			4		
18	3			5		
19	1	3				
20	5					4
21	3			6		
22	2		5			
23	4				5	
24	1	4				
25	3			7		
26	4				6	
27	3			8		
28	5					5
29	1	5				
30	3			9		
31						



GROUP TASK
Discussion

Describe the purpose and data type of Count, Item and Total in regard to this algorithm.



GROUP TASK
Discussion

Discuss the similarities between this algorithm and a standard linear search algorithm.

BINARY SEARCH

The word binary means two. A binary search involves splitting the list into two parts each time a comparison is made. As the list must have been previously sorted, one half of the list can be discarded after each comparison is made. Eventually the required item will be found. A list of items containing 100,000 items will after 1 comparison be reduced to 50,000 items, after two comparisons to 25,000 items, 3 comparisons 12,500 items, and so on. For 100,000 items it will take a maximum of only 17 comparisons to locate a specific item in the list. As $2^{16} = 65{,}536$ and $2^{17} = 131{,}072$, and there are 100,000 items, and 100,000 lies between 2^{16} and 2^{17}, the maximum number of comparisons is 17. In general, a list containing X items will take $\ln X/\ln 2$ (rounded up to the nearest integer) comparisons to find any particular item. Binary search is a O(log n) algorithm and is therefore significantly more efficient than a linear search algorithm. However for a binary search the data must be sorted, so if this is not already true or is not practical then a linear search may be a better solution.



The concept

The list of data items must be sorted. The middle item in the list is compared to the required item. If the required item lies in the first half of the list then the second half is discarded. If the required item lies in the second half of the list then the first half is discarded. This process is repeated with the remaining list of items. Eventually the required item will be found or the list of possible items will be empty.

```
BEGIN BinarySearch
    Set Low to 1
    Set High to number of items in list
    Set Found to False
    Get ItemToFind
    WHILE High >= Low AND Found = False
        Set Middle to INT((Low + High)/2)
        IF ItemToFind < Item(Middle) THEN
            Set High to Middle - 1
        ELSEIF ItemToFind = Item(Middle) THEN
            Set Found to True
        ELSE
            Set Low to Middle + 1
        ENDIF
    ENDWHILE
    IF Found = True THEN
        Display "Found"
    ELSE
        Display "Not found"
    ENDIF
END BinarySearch
```

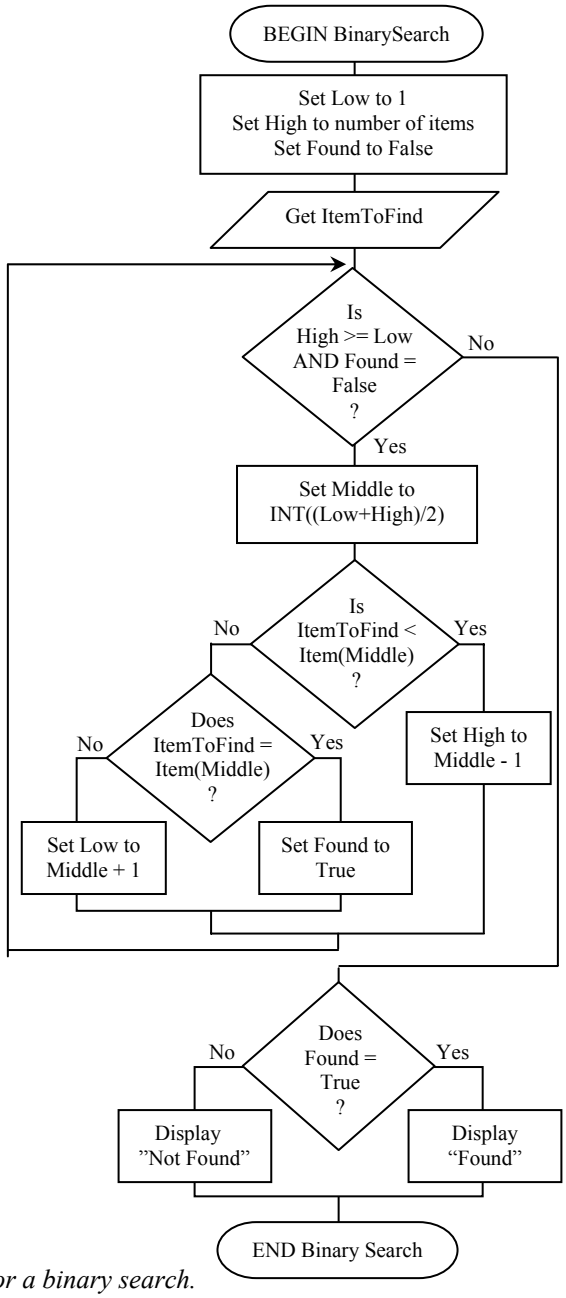


Fig 4.7
Pseudocode and flowchart for a binary search.



Consider the following:

A list of animals has been sorted into alphabetical order. The list is stored in an array called Animals. A binary search is required to find if a particular animal already exists within the array Animals.

Let us consider the following sample Animals array. Using the binary search algorithm from the previous page, with the Items array replaced by the Animals array, we would first set Low to 1 and High to 10. Found is also set to False.

1	2	3	4	5	6	7	8	9	10
Ant	Beetle	Cow	Dog	Frog	Goose	Hen	Moose	Rat	Yak
Low									High

We now get an ItemToFind, say Cow. As High is greater than Low and Found is False we progress into the body of the loop and set Middle to INT((1 + 10)/2), which is 5.

1	2	3	4	5	6	7	8	9	10
Ant	Beetle	Cow	Dog	Frog	Goose	Hen	Moose	Rat	Yak
Low				Middle					High

We now compare the required item Cow with the data item associated with 5, namely Frog. As Cow is less than Frog, in an alphabetical sense, High is set to 4, the current value of Middle - 1. In essence, we have discarded the top half of the array.

1	2	3	4
Ant	Beetle	Cow	Dog
Low			High



Flag
A variable used to indicate that a certain condition has been met. Usually a Flag is set to either True or False. In the binary search algorithm, Found is a Flag.

We now repeat the process. Middle is set to INT((1 + 4)/2), which is 2.

1	2	3	4
Ant	Beetle	Cow	Dog
Low	Middle		High

Cow is compared to item Animals(2). It is not less than Beetle and it is not equal to Beetle, so Low is set to 3, the current value of Middle + 1. The lower half of the array is discarded.

3	4
Cow	Dog
Low	High

The process repeats again. Middle is set to INT((3 + 4)/2), which is 3. We compare the required item with Animals(3) and find we have a match. Found is set to True and we then drop out of the loop. As Found is True the message Found is displayed.



GROUP TASK Activity
Choose an animal that is not in the above array. Work through the binary search to ensure the algorithm operates correctly and the output is Not found.



GROUP TASK Activity
Write down a list of 30 numbers in order from lowest to highest. Select one of the numbers and perform a desk check of the binary search algorithm.



HSC style question:

Consider the following algorithm when answering question 1 and 2.

```
BEGIN
    Index=0
    Result = This(Index)
    REPEAT
        Increment Index
        IF Result > This(Index) THEN
            Result = This(Index)
        ENDIF
    UNTIL Index = 100
END
```

1. The above algorithm includes which of the following control structures?
(A) A pre-test loop and binary selection.
(B) A pre-test loop and multi-way selection.
(C) A post-test loop and binary selection.
(D) A post-test loop and multi-way selection.

2. Which of the following best describes the processing performed by the algorithm?
(A) A binary search to find the largest value.
(B) A binary search to find the smallest value.
(C) A linear search to find the largest value.
(D) A linear search to find the smallest value.

3. A binary search is being used to locate a data item within a sorted list containing 100 data items. How many times will the body of the loop execute to locate a data item (or determine it is not present in the list)?
(A) exactly 2 times
(B) less than 8 times
(C) exactly 7 times
(D) less than 100 times

Suggested Solutions

1. (C) **2.** (D) **3.** (B)

SET 4A

1. A search that involves comparing each item in turn with the required item is called a:
   (A) binary search.
   (B) descending search.
   (C) quadratic search.
   (D) linear search.
2. A sorted list is one of the requirements of a:
   (A) binary search.
   (B) descending search.
   (C) quadratic search.
   (D) linear search.
3. Finding the maximum data item in a list is relevant for:
   (A) numeric data.
   (B) string or text data.
   (C) a sorted list.
   (D) both A and B.
4. A search of a data list containing 20,000 items takes on average 10,000 comparisons to find a particular item. The type of search being used is probably a:
   (A) binary search.
   (B) minimum search.
   (C) maximum search.
   (D) linear search.
5. Functions and procedures are used to implement algorithms. Which of the following statements is true?
   (A) Procedures always return one value; functions can return any number of values.
   (B) Functions always return one value; procedures can return any number of values.
   (C) Functions are only able to process numeric data.
   (D) All functions must include at least one input parameter.
6. The process of progressively breaking a problem down into its component parts is known as:
   (A) algorithm development.
   (B) top-down design.
   (C) system modelling.
   (D) stepwise refinement.
7. A library of routines is:
   (A) used to store standard algorithms for reuse.
   (B) used to store standard modules of source code for reuse.
   (C) of use to end users who wish to modify the original product.
   (D) a common way of storing the top-down design of a program.
8. An algorithm description language:
   (A) provides excellent user documentation.
   (B) is either a flowchart or pseudocode.
   (C) provides a method of expressing algorithms.
   (D) is a type of programming language.
9. Any particular item is always found in a list of 1000 items within 10 comparisons. What can be said about the 1000 items?
   (A) There must be only 10 unique items in the list.
   (B) The list of items must be sorted.
   (C) Most of the list must be empty.
   (D) There is insufficient information to answer this question.
10. The largest value in an array of 10,000 items is found to be in position 234. Which of the following is True?
    (A) The list must be sorted.
    (B) Larger items may exist in positions 235 to 10,000.
    (C) The list must be unsorted.
    (D) The list only contains numeric data.

11. Often, with numeric decimal values, a problem will involve finding a value within a specific range. Design a linear search algorithm that will search for and display all the values in a list between a lower and upper value.
12. Create an algorithm to find and output both the maximum and the minimum values stored in an array of unsorted data items.
13. Perform a desk check of both the linear search algorithm and the binary search algorithm using the following test data: The Item array contains the 6 elements CPU, Keyboard, Monitor, Mouse, Printer and Scanner indexed from 1 to 6. The input to the algorithms is Printer.
14. An array contains a sorted list of 2000 people's surnames. Design an algorithm using a binary search strategy that will find the number of occurrences of a particular surname.
15. Your English teacher wishes you to develop a program to tally marks for them. The marks are to be tallied into deciles. For example, a mark of 72 is in the 8th decile, 36 is in the 4th decile and a mark of 6 is in the 1st decile. Design an algorithm to accomplish this task. Assume the marks are stored in an unsorted array called Marks, which is indexed from 0 to 99.

INSERTION SORT

Insertion sorts are used to arrange data into order. They are particularly useful when a small number of items need to be added to an already sorted list. Insertion sorts involve looking through a sorted list for the correct position for a new item and inserting the item into that position.

For example, imagine we are playing a card game. At this stage of the game we pick up two cards and add or insert them into our existing hand of sorted cards. We are using an insertion sort to accomplish this task.

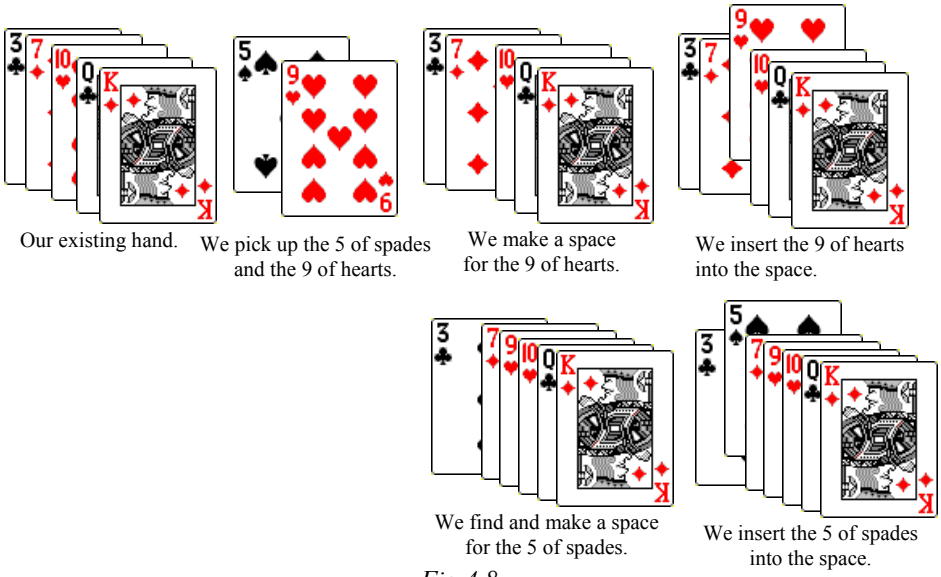


Fig 4.8
Cards inserted using an insertion sort strategy.



The simplified concept

An unsorted and a sorted list are maintained. Each item in the unsorted list is considered in turn. Its correct place in the sorted list is found. A space is made by moving (or shuffling) items up one place. Finally, the item is placed in position. This process continues until no items remain in the unsorted list.

```
BEGIN
   WHILE more items to insert
      Find correct position for item
      Make space for new item
      Insert item
   ENDWHILE
END
```

Fig 4.9
A simplified insertion sort algorithm.



GROUP TASK Practical Activity
Write about 20 different words on separate slips of paper. Use an insertion sort strategy to sort the words into alphabetical order.



GROUP TASK Discussion
Insertion sorts in Big O are $O(n^2)$. Based on the practical activity above, can you explain why this is the case?



The enhanced concept

The following algorithm expands on the simplified version above. In this version, a complete array called Item is sorted. We start by considering the first item to be the sorted part of the array. We then consider where the second item should be placed in relation to the first item, and insert it into position. The third item is then considered. Its correct place in the sorted list is found, a space is made and the item is inserted. This process continues until no items remain in the unsorted list.

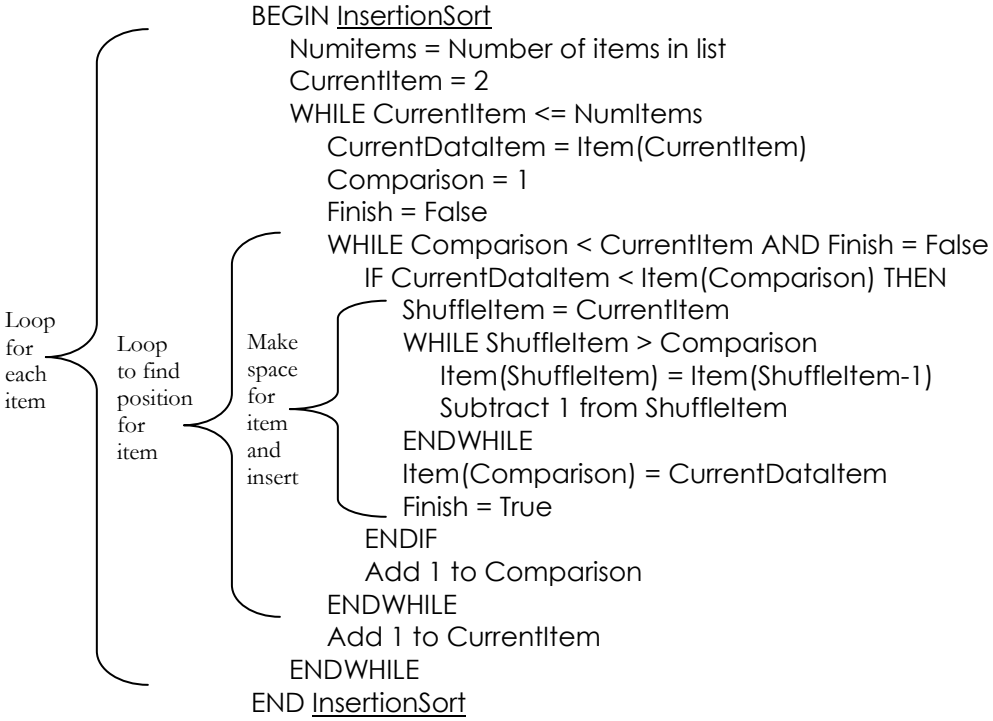


Fig 4.10
Expanded version of an insertion sort algorithm.



Consider the following:

The array of animals below is to be sorted using the insertion sort described in the algorithm in *Fig 4.10*. Let us examine the array after each item has been inserted. The sorted list initially contains only the first item, Goose, and we consider the second item, Yak.

1	2	3	4	5	6	7	8	9	10
Goose	Yak	Ant	Dog	Moose	Cow	Hen	Rat	Frog	Beetle

Yak is now in the sorted part of the list and we consider the third item, Ant:

1	2	3	4	5	6	7	8	9	10
Goose	Yak	Ant	Dog	Moose	Cow	Hen	Rat	Frog	Beetle

Ant belongs to the left of Goose, so Goose and Yak have been moved up one and Ant inserted into the space. We now consider the fourth item, Dog:

1	2	3	4	5	6	7	8	9	10
Ant	Goose	Yak	Dog	Moose	Cow	Hen	Rat	Frog	Beetle

Dog belongs between Ant and Goose, so Goose and Yak are moved to the right and Dog is inserted. We now consider the fifth item, Moose:

1	2	3	4	5	6	7	8	9	10
Ant	Dog	Goose	Yak	Moose	Cow	Hen	Rat	Frog	Beetle

Moose has been inserted into position. We now consider Cow:

1	2	3	4	5	6	7	8	9	10
Ant	Dog	Goose	Moose	Yak	Cow	Hen	Rat	Frog	Beetle

Cow is inserted between Ant and Dog. We now consider the seventh item, Hen:

1	2	3	4	5	6	7	8	9	10
Ant	Cow	Dog	Goose	Moose	Yak	Hen	Rat	Frog	Beetle

Space is made for Hen in position five and Hen is inserted. We now consider Rat:

1	2	3	4	5	6	7	8	9	10
Ant	Cow	Dog	Goose	Hen	Moose	Yak	Rat	Frog	Beetle

Rat comes before Yak, so Yak is shuffled up and Rat inserted. We now examine Frog:

1	2	3	4	5	6	7	8	9	10
Ant	Cow	Dog	Goose	Hen	Moose	Rat	Yak	Frog	Beetle

Frog is inserted into position four by shuffling up items. Finally, Beetle is examined:

1	2	3	4	5	6	7	8	9	10
Ant	Cow	Dog	Frog	Goose	Hen	Moose	Rat	Yak	Beetle

Space is made and Beetle is inserted. As there are no more items in the unsorted part of the list, the sort is complete:

1	2	3	4	5	6	7	8	9	10
Ant	Beetle	Cow	Dog	Frog	Goose	Hen	Moose	Rat	Yak



GROUP TASK Discussion

The insertion sort described in *Fig 4.10* is used to sort an entire array of data. Is this the most appropriate sort for this purpose? Discuss.



GROUP TASK Discussion

Perform an insertion sort showing the state of the array after each item has been inserted, using the following data:

34, 44, 23, 35, 34, 55, 42, 12, 52, 16, 16, 17, 56, 78, 26, 23, 41, 29, 30, 42.

This time perform the sort from right to left. That is, initially the sorted part of the array will contain 42, and the first item to be inserted will be 30.

SELECTION SORT

Selection sorts essentially are a repetition of a linear search to find the smallest (or largest) item in a list. Each time the search is complete, the item found is moved to the end of the sorted list. This type of sort is rather more intuitive for humans as it is a strategy often employed when sorting things by hand.

For example, when playing card games we often arrange the cards in our hand using this selection sort method.

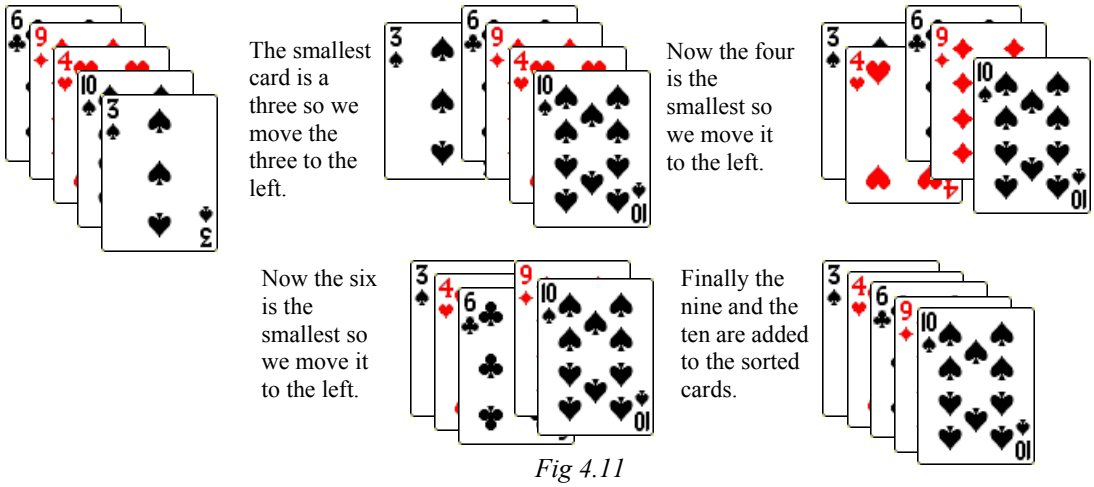


Fig 4.11
Cards sorted using a selection sort strategy.



The simplified concept

A sorted and an unsorted list are maintained. The smallest item in the unsorted list is swapped with the first item in the unsorted list. The sorted list is increased in length to include the new item. This process is repeated while there are items remaining in the unsorted list.

```
BEGIN SimpleSelectionSort
    WHILE unsorted list has items
        Find smallest item
        Swap with first item in unsorted list
        Increase sorted list size by 1
    ENDWHILE
END SimpleSelectionSort
```

Fig 4.12
A simplified selection sort algorithm.



Swap
A swap is used to exchange the content of one variable with that of another. A temporary variable and three steps are generally required.
Temp = X
X = Y
Y = Temp



GROUP TASK Practical Activity
Write about 20 different words on separate slips of paper. Use a selection sort strategy to sort the words into alphabetical order.



GROUP TASK Discussion
Selection sorts in Big O are $O(n^2)$. Based on the practical activity above, can you explain why this is the case?



The enhanced concept

The list is examined one item at a time for the smallest item. This item is swapped with the first item in the list. This first item is now sorted and becomes the sorted list. The remaining unsorted list is examined for the smallest item. This item is swapped with the first item in the unsorted list. The length of the sorted list is increased by one. There are now two items in the sorted list. This process is repeated while there are still items remaining in the unsorted list.

```
BEGIN SelectionSort
   Pass = 1
   WHILE Pass < Number of items
      Count = Pass + 1
      Minimum = Pass
      WHILE Count <= Number of Items
         IF Item(Count) < Item(Minimum) THEN
            Minimum = Count
         ENDIF
         Count = Count + 1
      ENDWHILE
      Swap Item(Minimum) with Item(Pass)
      Pass = Pass + 1
   ENDWHILE
END SelectionSort
```

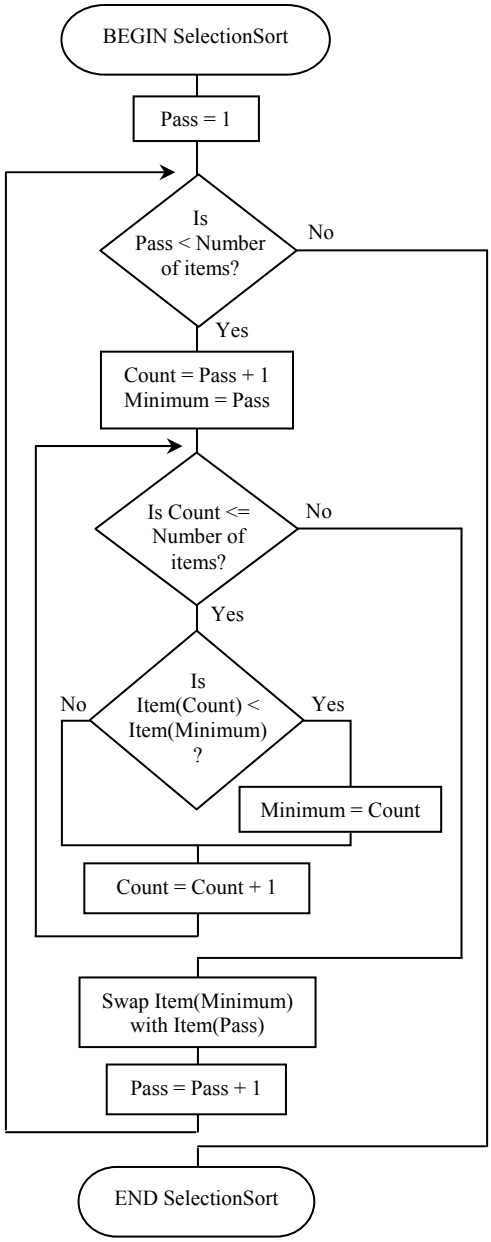


Fig 4.13
Pseudocode and flowchart for a selection sort.



Consider the following:

The array of animals below is to be sorted using the selection sort described in the algorithm in *Fig 4.13*. Firstly, we search for the smallest item in the list, namely Ant:

1	2	3	4	5	6	7	8	9	10
Goose	Yak	Ant	Dog	Moose	Cow	Hen	Rat	Frog	Beetle

Ant is swapped with the first item, Goose. The new first item, Ant, now forms the sorted part of the array. The unsorted list is searched for the smallest item, namely Beetle:

1	2	3	4	5	6	7	8	9	10
Ant	Yak	Goose	Dog	Moose	Cow	Hen	Rat	Frog	Beetle

Beetle is swapped with the first item in the unsorted list, namely Yak. The unsorted list is again scanned for the smallest item, and Cow is found:

1	2	3	4	5	6	7	8	9	10
Ant	Beetle	Goose	Dog	Moose	Cow	Hen	Rat	Frog	Yak

Cow and Goose are swapped. The smallest item is now Dog:

1	2	3	4	5	6	7	8	9	10
Ant	Beetle	Cow	Dog	Moose	Goose	Hen	Rat	Frog	Yak

Dog is swapped with itself. The smallest item is now Frog:

1	2	3	4	5	6	7	8	9	10
Ant	Beetle	Cow	Dog	Moose	Goose	Hen	Rat	Frog	Yak

Frog and Moose are swapped. Goose is now the smallest item:

1	2	3	4	5	6	7	8	9	10
Ant	Beetle	Cow	Dog	Frog	Goose	Hen	Rat	Moose	Yak

Goose is swapped with itself. Hen is now the smallest item:

1	2	3	4	5	6	7	8	9	10
Ant	Beetle	Cow	Dog	Frog	Goose	Hen	Rat	Moose	Yak

Hen is swapped with itself. Moose is now the smallest item:

1	2	3	4	5	6	7	8	9	10
Ant	Beetle	Cow	Dog	Frog	Goose	Hen	Rat	Moose	Yak

Moose is swapped with Rat. Rat is now the smallest item:

1	2	3	4	5	6	7	8	9	10
Ant	Beetle	Cow	Dog	Frog	Goose	Hen	Moose	Rat	Yak

Rat is swapped with itself. As only one item remains, the list is now sorted.

1	2	3	4	5	6	7	8	9	10
Ant	Beetle	Cow	Dog	Frog	Goose	Hen	Moose	Rat	Yak



GROUP TASK Discussion

Examine the selection sort algorithm and describe how the variables Pass and Count combine to change the length of the sorted and unsorted sections of the array Item.



GROUP TASK Activity

Perform a selection sort showing the state of the array after a new item has been placed into the sorted part of the array, using the following data:

34, 44, 23, 35, 34, 55, 42, 12, 52, 16, 16, 17, 56, 78, 26, 23, 41, 29, 30, 42.

This time perform the sort from right to left. That is, search for the largest item each time and swap it with the right-hand item in the unsorted part of the array.

BUBBLE SORT

Bubble sorts, just like the insertion and selection sorts, are used to arrange data into either numerical or alphabetical order. Data items tend to bubble, or float to the end of the list when a bubble sort is undertaken. The list is sorted by considering the order of individual item pairs and swapping them if they are out of order. This results in items moving one position at a time towards their final position.



The simplified concept

Pairs of adjacent items are compared in turn. If they are out of order, they are swapped. Once the end of the list is reached, we move back to the start and repeat the process. Eventually no swaps will be required, indicating that the list is sorted.

```
BEGIN SimpleBubbleSort
   Swapped = True
   WHILE Swapped = True
      Swapped = False
      Comparison = 1
      WHILE Comparison<Number of items
         IF Item(Comparison)<Item(Comparison + 1)
            Swap Items
            Swapped = True
         ENDIF
         Add 1 to Comparison
      ENDWHILE
   ENDWHILE
END SimpleBubbleSort
```

Fig 4.14
Pseudocode for a simple bubble sort.

The Boolean variable Swapped is used as a flag to determine if a swap has occurred during a pass through the array Items. If no swap has occurred, indicating that the array is sorted, then Swapped will have a value of False and the algorithm will end. The variable Comparison is used to traverse the array. For example, when Comparison is equal to 5, items 5 and 6 are being compared; when Comparison equals 6, items 6 and 7 are being compared.



Boolean Variable
A variable that can hold one of two values: True or False. Boolean variables are often used as flags to indicate that a certain condition has been met.



GROUP TASK Practical Activity
Write about 20 different words on separate slips of paper. Use a bubble sort strategy to sort the words into alphabetical order.



GROUP TASK Discussion
Bubble sorts in Big O are $O(n^2)$. Based on the practical activity above, can you explain why this is the case?



Consider the following:

The simple bubble sort described in the algorithms in *Fig 4.14* and *Fig 4.15* will correctly sort an array of items into ascending order.



GROUP TASK
Activity

Rewrite the simple bubble sort algorithm so that the array is sorted into descending order.



GROUP TASK
Activity

Using the following list of data items, complete a full desk check of the simple bubble sort algorithm:

34, 44, 23, 35, 34, 55, 42, 12, 52, 16, 16, 17, 56, 78, 26, 23, 41, 29, 30, 42.

Perform a desk check of your descending bubble sort algorithm using the above data list.

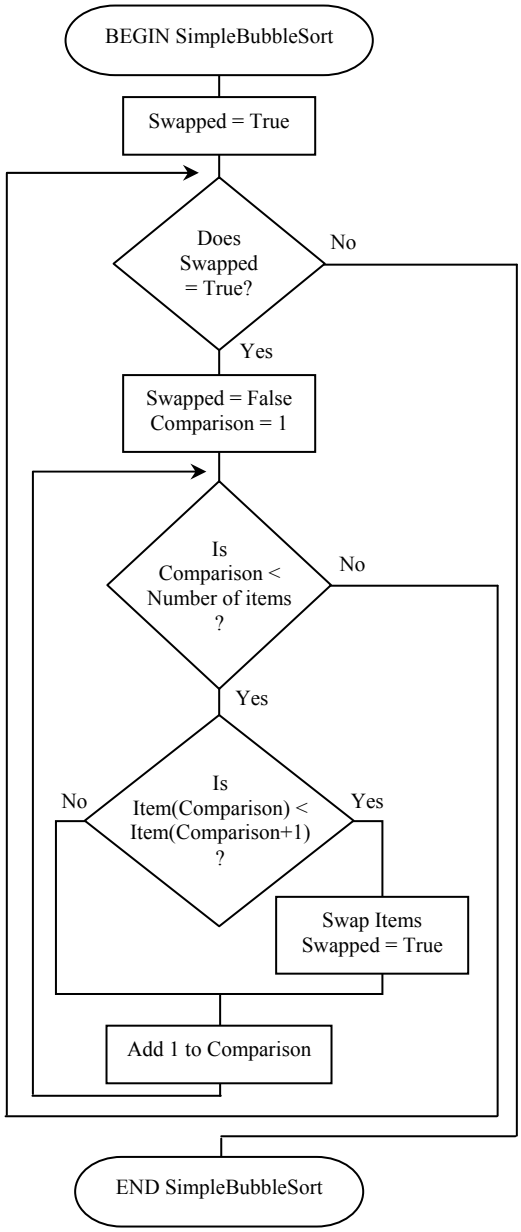


Fig 4.15
Flowchart for a simple bubble sort.

The simple bubble sort can be further enhanced to optimise its operation. We will look at an enhanced bubble sort algorithm in the next section. Before considering the enhanced bubble sort it is important that you be confident in the operation of the simplified version. Complete a number of desk checks of the algorithm to become confident in regard to its operation.



The enhanced concept

The enhanced bubble sort recognises that after each pass through the list the largest item will be in its correct position at the right-hand end of the list. It is therefore not necessary to consider this item in future passes through the list. In other words, the length of the unsorted list always decreases by one after each pass. If we keep track of the number of passes completed, we can then just complete comparisons up to the number of items in the list minus the number of completed passes.

Let us consider the saving in terms of comparisons. For a list of 100 items, using the simplified version could take 99 passes through the list, and each pass involves 99 comparisons and possible swaps. Ninety-nine times 99 results in 9,801 comparisons necessary to be sure the list is sorted. The enhanced bubble sort could still take 99 passes to complete, however the first pass will involve 99 comparisons, the second 98 comparisons, the third 97 and so on. The last pass will involve only one comparison as there will only be 2 items remaining in the list. The average of 99, 98, 97…3, 2, 1 is 50 - that is, the average number of comparisons per pass is 50. Ninety-nine passes times 50 comparisons per pass results in a total of 4,950 comparisons. This is approximately half the number required by the simplified bubble sort. The larger the list of data items, the larger the saving in comparisons. In general, the number of comparisons is halved using the enhanced version rather than the simplified version. Although halving the processing is a significant improvement in terms of Big O notation the bubble sort is still classified as $O(n^2)$ as it is not even remotely close to the performance of an $O(n \log n)$ sort algorithm.

```
BEGIN EnhancedBubbleSort
   Swapped = True
   Pass = 0
   WHILE Swapped = True
      Swapped = False
      Comparison = 1
      WHILE Comparison<Number of items – Pass
         IF Item(Comparison)<Item(Comparison + 1)
            Swap Item(Comparison) with Item(Comparison + 1)
            Swapped = True
         ENDIF
         Add 1 to Comparison
      ENDWHILE
      Add 1 to Pass
   ENDWHILE
END EnhancedBubbleSort
```

Fig 4.16
Pseudocode for the enhanced bubble sort.



Consider the following:

The array of animals below is to be sorted using the enhanced bubble sort described in the algorithm in *Fig 4.14*. Let us examine the array after each pass is completed.

1	2	3	4	5	6	7	8	9	10
Goose	Yak	Ant	Dog	Moose	Cow	Hen	Rat	Frog	Beetle

After the first pass, with Yak now in its correct position:

1	2	3	4	5	6	7	8	9	10
Goose	Ant	Dog	Moose	Cow	Hen	Rat	Frog	Beetle	Yak

After the second pass. Notice that Goose and Moose are bubbling up:

1	2	3	4	5	6	7	8	9	10
Ant	Dog	Goose	Cow	Hen	Moose	Frog	Beetle	Rat	Yak

After the third pass, Goose has moved up and Moose is now correct:

1	2	3	4	5	6	7	8	9	10
Ant	Dog	Cow	Goose	Hen	Frog	Beetle	Moose	Rat	Yak

After the fourth pass, Hen is now correct:

1	2	3	4	5	6	7	8	9	10
Ant	Cow	Dog	Goose	Frog	Beetle	Hen	Moose	Rat	Yak

After the fifth pass, five items are now in their correct positions:

1	2	3	4	5	6	7	8	9	10
Ant	Cow	Dog	Frog	Beetle	Goose	Hen	Moose	Rat	Yak

After the sixth pass:

1	2	3	4	5	6	7	8	9	10
Ant	Cow	Dog	Beetle	Frog	Goose	Hen	Moose	Rat	Yak

After the seventh pass:

1	2	3	4	5	6	7	8	9	10
Ant	Cow	Beetle	Dog	Frog	Goose	Hen	Moose	Rat	Yak

After the eighth pass:

1	2	3	4	5	6	7	8	9	10
Ant	Beetle	Cow	Dog	Frog	Goose	Hen	Moose	Rat	Yak

The ninth pass completes the process:

1	2	3	4	5	6	7	8	9	10
Ant	Beetle	Cow	Dog	Frog	Goose	Hen	Moose	Rat	Yak



GROUP TASK Discussion
Examine each pass in the above sort. How many swaps have occurred during each pass?



GROUP TASK Discussion
In this example, no swap occurred on the ninth pass. If a swap had occurred on the ninth pass, explain how the sort would have ended. Examine the value of any relevant variables as part of your discussion.



GROUP TASK Research
Research, using the internet or otherwise, other sorting algorithms which are more efficient than insertion, selection or bubble sorts.



HSC style questions:

1. To maximise the number of vehicles that can be loaded on a passenger/car ferry the following strategy is used:

I. Passengers park their vehicles in a large parking lot, leaving their keys in the vehicle.

II. Employees choose the largest vehicle from the parking lot and drive it onto the ferry.

III. Step II is repeated until all cars have been loaded.

The above strategy is an example of?

(A) An insertion sort.

(B) A selection sort.

(C) A bubble sort.

(D) A linear search.

Consider the following algorithm when answering questions 2 and 3.

```
1   BEGIN SortSwap(arrItems, Index)
2       IF arrItems[Index] < arrItems[Index + 1] THEN
3           Temp = arrItems[Index]
4           arrItems[Index + 1] = arrItems[Index]
5           arrItems[Index + 1] = Temp
6       ENDIF
7   END SortSwap
```

2. The algorithm incorrectly swaps two items. Which modification corrects the error?

(A) Exchange lines 3 and 5.

(B) Exchange lines 4 and 5.

(C) Change line 5 to arrItems[Index] = Temp

(D) Change line 4 to arrItems[Index] = arrItems[Index + 1]

3. The corrected algorithm would be useful as part of which sort algorithm?

(A) A descending selection sort.

(B) An ascending selection sort.

(C) A descending bubble sort.

(D) An ascending bubble sort.

Question 4.

Complete the table to show the arrangement of elements after each pass of an ascending insertion sort.

Dog	Cat	Goat	Ape	Cow	Frog	Bird

Suggested Solutions

1. (B)

2. (D)

3. (C)

Question 4.

Sorting from left to right:

Dog	Cat	Goat	Ape	Cow	Frog	Bird
Cat	**Dog**	Goat	Ape	Cow	Frog	Bird
Cat	**Dog**	**Goat**	Ape	Cow	Frog	Bird
Ape	**Cat**	**Dog**	**Goat**	Cow	Frog	Bird
Ape	**Cat**	**Cow**	**Dog**	**Goat**	Frog	Bird
Ape	**Cat**	**Cow**	**Dog**	**Frog**	**Goat**	Bird
Ape	**Bird**	**Cat**	**Cow**	**Dog**	**Frog**	**Goat**

OR

Sorting from right to Left:

Dog	Cat	Goat	Ape	Cow	Frog	Bird
Cat	Dog	Goat	Ape	Cow	**Bird**	**Frog**
Cat	Dog	Goat	Ape	**Bird**	**Cow**	**Frog**
Cat	Dog	Goat	**Ape**	**Bird**	**Cow**	**Frog**
Cat	Dog	**Ape**	**Bird**	**Cow**	**Frog**	**Goat**
Cat	**Ape**	**Bird**	**Cow**	**Dog**	**Frog**	**Goat**
Ape	**Bird**	**Cat**	**Cow**	**Dog**	**Frog**	**Goat**



GROUP TASK Discussion
Question 1 is a practical implementation of a selection sort. Brainstorm similar practical implementations for an insertion sort.



GROUP TASK Discussion
Answer question 4 again using a selection sort and then using a bubble sort strategy.

SET 4B

1. Adding a small number of items to an existing sorted list is best accomplished using:
   (A) a bubble sort.
   (B) a selection sort.
   (C) an insertion sort.
   (D) a pack of cards.

2. Continuously finding the smallest (or largest) item in a list is the major concept in:
   (A) a bubble sort.
   (B) a selection sort.
   (C) an insertion sort.
   (D) a binary search.

3. Swapping pairs of out of order items occurs during:
   (A) a bubble sort.
   (B) a selection sort.
   (C) an insertion sort.
   (D) a quick sort.

4. During which sort(s) does a sorted list grow at one end of the array?
   (A) a bubble sort.
   (B) a selection sort.
   (C) an insertion sort.
   (D) a selection and insertion sort.

5. An ascending selection sort could be performed using which strategy during each pass?
   (A) Find the largest unsorted item and swap with the last unsorted item.
   (B) Find the smallest unsorted item and swap with the first unsorted item.
   (C) Find the smallest unsorted item and swap with the last unsorted item.
   (D) Either (A) or (B).

6. In terms of Big 0, which of the following sorts are $O(n^2)$?
   (A) bubble sort
   (B) selection sort
   (C) insertion sort
   (D) All of the above

7. Which of the following shows the integers 7364182 after 3 passes of a descending insertion sort?
   (A) 3467182
   (B) 3674182
   (C) 7634182
   (D) 7643182

8. The two assignment statements X=Y and Y=X would result in:
   (A) the contents of X and Y being swapped.
   (B) both X and Y containing the original value of X.
   (C) both X and Y containing the original value of Y.
   (D) no change to the values held in X and Y.

9. Variables that can hold either the value True or the value False are known as:
   (A) flags.
   (B) arrays.
   (C) numeric variables.
   (D) Boolean variables.

10. An array is being sorted. After examining each item in the array the largest item is known to be in its correct position. This sort could be a(n):
    (A) insertion or bubble sort.
    (B) insertion or selection sort.
    (C) selection or bubble sort.
    (D) insertion, selection or bubble sort.

11. Make a list of all the students in your roll call class. Cut up the list so that each person's name is on a separate slip of paper. Shuffle the slips of paper. Sort these slips using each of the three sorting strategies studied in this chapter: that is, first sort using an insertion sort, then shuffle again and sort using a selection sort, and finally repeat the process using a bubble sort.

12. What changes need to be made to the insertion sort in *Fig 4.10* so that the array will be sorted into descending order?

13. In your own words describe the sorting strategy employed when using an insertion sort, a selection sort, and a bubble sort.

14. Research at least two other sorting algorithms and outline the logic. Try to choose sorts which perform better than $0(n^2)$.

15. An existing array contains a sorted list of all of a company's clients' names. Every day or so, a new client needs to be added to this array in the correct position. Develop an algorithm that will accomplish this task.

PROCESSING STRINGS

Strings are data types used to hold characters or text. Many applications require the manipulation of string data as part of their solution. A string is essentially an array of characters; in some programming languages strings must be declared as arrays of characters. Some useful routines for processing strings include ones for extracting, deleting or inserting strings out of or into other strings. Most programming languages include functions for performing these activities. In this section, we examine the algorithms behind these standard string manipulation functions.

Joining two strings is known as concatenation. Most programming languages use either an ampersand (&) or an addition sign (+) as the symbol for concatenation. In this text we will use the addition sign to indicate concatenation. For example, "The" + "Cat" would result in the string "TheCat". If the string variable Word contains "Dogs" and the variable Phrase contains the string "chase balls", then Word + " " + Phrase results in the concatenated string "Dogs chase balls".

Let us now consider algorithms for extracting, deleting and inserting strings out of and into other strings. These algorithms have been written as functions with inputs via parameters, as this is generally the way they are implemented in most programming languages.



Parameter
A value or variable passed to or returned from a function or procedure. Parameters are often called arguments.

Extracting

Extracting data from a string is essentially making a copy of a section of the original string; the original string remains intact. Three pieces of information are required to achieve the extraction: the original or initial string; the starting point for the extraction; and either the length of the extraction string or the finish point of the extraction string.

The algorithm in *Fig 4.17* describes a function that receives the start and finish positions of the required string, together with the actual initial string, as input via the three parameters Start, Finish and ExtractString. The characters within the initial string from position Start to position Finish are returned by the function.

```
BEGIN Extract(Start,Finish,ExtractString)
   Temp = Null string
   Position = Start
   WHILE Position <= Finish
      Temp = Temp + ExtractString(Position)
      Add 1 to Position
   ENDWHILE
   RETURN Temp
END Extract
```

Fig 4.17
Extract algorithm for strings.

The function call:

Extract(6, 8, "Mathematics") returns the string "mat".



GROUP TASK Investigation
Will the Extract algorithm work correctly for all possible inputs? Explain how the inputs could be validated to avoid errors.



GROUP TASK Activity
Design an algorithm for an extract function using as a parameter the length of the required string, rather than the finish position.

Deleting

Deleting data from a string is essentially cutting a portion from the string. The original string is reduced in length by the number of characters that have been removed. As for the extraction algorithm, three identical parameters are required: the original or initial string; the starting point for the deletion; and either the length of the string to be deleted or the finish point of the string to be deleted.

The deletion algorithm can be created using the extract function (see *Fig 4.18)*. This algorithm takes the first part of the array up to the characters to be deleted and concatenates this string with the characters that follow the string to be deleted.

```
BEGIN Delete(Start,Finish,DeleteString)
   Temp = Extract(1,Start-1,DeleteString)
   Length = Length of DeleteString
   Temp = Temp + Extract(Finish+1,Length,DeleteString)
   RETURN Temp
END Delete
```

Fig 4.18
Delete algorithm using the Extract function.

A more efficient algorithm, such as the one in *Fig 4.19*, moves each character occuring after the characters to be deleted forward to join the existing front of the original array. The length of the string is then reduced by the length of the deleted string.

```
BEGIN Delete(Start,Finish,DeleteString)
   Temp = DeleteString
   Position = Start
   Length = Finish – Start + 1
   StrLength = length of DeleteString - Length
   WHILE Position <= StrLength
      Temp(Position) = Temp(Position + Length)
      Add 1 to Position
   ENDWHILE
   Reduce size of Temp by Length
   RETURN Temp
END Delete
```

Fig 4.19
Delete algorithm for strings.



GROUP TASK Activity
Design an algorithm for a delete function using as a parameter the length of the required string, rather than the finish position.

Inserting

Inserting a string of characters into another string of characters involves splitting the initial string into two parts and then concatenating the first part of the string with the insertion string and the final part of the initial string. Parameters required to complete this process are the start position for the insertion, the insertion string, and the main or initial string.

This process can be accomplished by making use of the extract function algorithm, as shown in *Fig 4.20*. This algorithm first sets Temp to the first part of MainString up to position Start. To this is added the InsertString, and then the remainder of MainString is concatenated with the variable Temp. As this is a function, the function returnd the string contained in Temp. For example, Insert(7,"lovely ","Hello people") returns "Hello lovely people".

```
BEGIN Insert(Start,InsertString,MainString)
   Temp = Extract(1,Start-1,MainString)
   Temp = Temp + InsertString
   Length = Length of MainString
   Temp = Temp + Extract(Start,Length,MainString)
   RETURN Temp
END Insert
```

Fig 4.20
Insert algorithm using the Extract function.



GROUP TASK Activity
Desk check the insert algorithm in *Fig 4.20* using the following call:
Insert(16, "not ", "Algorithms are hard")

RANDOM NUMBERS

Generating Random Numbers

Most high level languages include the ability to generate random numbers. Many include a class called Random which is able to generate random integers and also random floating point numbers. The syntax of these functions or methods will vary depending on the language. In most cases the numbers generated are pseudo-random numbers, which are generated using an initial seed into a mathematical formula. Although these pseudo-random numbers are suitable for most purposes they should not be used for cryptographic or gambling applications where true randomness is required. For such applications random numbers created by sampling some real world analog value are commonly used.

When generating integers it is often possible to specify a range. In .NET the Random class includes a next method which takes a minimum and maximum value. The integer generated is greater than or equal to the minimum value and less than the maximum value. If ran is an instance of Random then in .NET ran.Next(4,6) generates either 4 or 5. In Java only a maximum value can be provided and the minimum is always 0. So in Java ran.nextInt(6) generates 0, 1, 2, 3, 4 or 5. PHP includes the rand() function. In PHP rand(4,6) generates 4, 5, or 6. PHP, Java and .NET all implement their random functions differently, so ensure you read the documentation carefully for the random number generator you use.

In many languages, including both .NET and Java random floating point numbers are generated from 0 up to but not including 1. Other languages do not include random float generators, in which case it is simple to create one using the random integer generator divided by the maximum integer value. For example, in PHP rand(0) / RAND_MAX generates a random float from 0 up to but not including 1. Random float values can then be scaled and translated to suit the required range. For instance in Java and .NET to generate a random floating point number from 3 up to 10 the following code could be used r = ran.nextDouble() * 7 + 3 or in general terms r = ran.nextDouble() * (max - min) + min.

Older versions of BASIC use RND to generate a float from 0 up to but not including 1 – statements to generate random integers are not included. In these languages the random number is scaled, converted to an integer and then translated. For example to generate the integers 1 to 6 inclusive the BASIC statement r = INT(RND * 6) +1 is used. RND * 6 generates a number from 0 up to 5.99..., the INT function simply removes the decimal or fractional part leaving an integer in the range 0 to 5 and finally adding 1 translates our range to be 1 to 6. In general terms r = INT(RND * (max - min +1)) + min.

When including random numbers within algorithms it is important to specify the data type (integer or floating point) as well as the range of values. Be careful to specify whether the range includes the end values or not.



GROUP TASK Discussion
Scaling terms within the general formulas given aboveare different. The float version uses max -min and then the BASIC integer version uses the term max - min +1. Why is this? Explain using examples.

Generating a Set of Unique Random Numbers

Many games utilise random numbers in a variety of ways. Obvious examples include shuffling cards, rolling dice and selecting lotto numbers, however they are also used in various less obvious ways such as moving background characters or creating realistic water or cloud movement.

Let us work through the process of creating and refining a function which will generate a set of unique random integers. The function will accept three parameters namely, the minimum value and the maximum value to be generated, and the number of unique values to generate. The function will return an array containing the generated random integer values. For instance the call randomArray(3, 6, 2) would return an array containing 2 different numbers from the set of possible numbers 3, 4, 5, 6.

Our first attempt uses the following logic:

1. Generate a random number, say r
2. Search through the array of random numbers already generated looking for r.
3. If we find r in the array then go back to step 1 to try again.
4. If we don't find r then add r to the array of random numbers.
5. Repeat steps 1 to 4 above until we have the required number of random numbers.

An algorithm which refines the above logic is shown in *Fig 4.21*. Notice there are a total of three nested loops (which should raise efficiency alarm bells). The outside FOR loop spins for each new unique random number, the REPEAT loop spins around until we generate a random number that is not already in the array and the inner WHILE loop does the actual search to check if the current r already exists in the array.

```
BEGIN randArray(min, max, num)
    FOR i = 0 TO num - 1
        REPEAT
            r = random integer from min to max inclusive
            c = 0
            found = False
            WHILE c < i AND NOT found
                found = arr(c) = r
                c = c + 1
            END WHILE
        UNTIL NOT found
        arr(i) = r
    NEXT i
    RETURN arr
END randArray
```

Fig 4.21
First attempt randArray algorithm.

Consider the performance of the algorithm in *Fig 4.21* in terms of Big O notation. The outside FOR loop goes around num times, so O(n). Now the REPEAT loop, worse case is it continues endlessly generating r values that are already in the array arr, so it's at least as bad as O(n). So far the algorithm is O(n^2), but we have another inner loop. This WHILE loop searches through all existing arr values. At the beginning of processing there are no values to search through and then at the end there are close to num values to search. On average about num/2 values will be searched, but in terms of Big O this is still close to O(n). Therefore this algorithm looks to be at best O(n^3), very poor performance indeed.



GROUP TASK Discussion
Before reading further, brainstorm improvements or alternatives to the algorithm in *Fig 4.21* which you consider may improve performance.

An obvious issue in our first algorithm is the time taken to check if we have already generated a number beforehand. Perhaps we can find a more efficient strategy for performing this check. One idea is to record which numbers have already been generated as they are generated. This is idea is much like maintaining a list of all the possible numbers in the range and then crossing them off as they are generated. Each time a number is generated we check the list to ensure the number has not yet been crossed off. If it hasn't then we cross it off and keep it in our array.

Our second attempt uses the following logic:

1. Create an array called used of Boolean values. The array used contains one element for each possible number that can be generated and all initialised to false.
2. Generate a random number, say r
3. If used(r) is true then we've already got this value so go back to step 2.
3. If used(r) is false then set it to true and add r to the array of random numbers.
4. Repeat 2 and 3 above until we have the required number of random numbers.

An algorithm using the above logic is shown in *Fig 4.22*. This algorithm has three loops, however only two are nested so these will have by far the greatest affect on performance. The second FOR loop spins for each number generated so O(n). The nested REPEAT loop spins until we generate a random number that hasn't been used. As in the last algorithm this could go on forever, so it's at least as bad as O(n). Overall our second attempt is at best an O(n^2) algorithm. A vast improvement on our first attempt, but can we do better?

```
BEGIN randArray(min, max, num)
    FOR i = 0 TO max - min
        used(i) = False
    NEXT i
    FOR i = 0 TO num - 1
        REPEAT
            r = random integer from min to max inclusive
        UNTIL used(r) = False
        used(r) = True
        arr(i) = r + min
    NEXT i
    RETURN arr
END randArray
```

Fig 4.22
Second attempt randArray algorithm.

The whole strategy of checking if a number has been used is flawed and unnecessary. Consider a pack of cards. Each card in the pack is already unique. So if we draw cards at random from the pack we are sure to get unique cards. Say we wish to generate 5 unique numbers from 3 to 10. We write the numbers 3 through to 10 on blank cards and randomly draw 5 cards as our 5 unique random numbers. We are guaranteed to have unique numbers as the card values were unique to start with. We'll use the same strategy in our third attempt at the randArray algorithm. However as we are using an array we'll need to be clever and overwrite used cards with ones from the end of the list so they're out of the way and we don't accidently use them again.

Our third attempt uses the following logic:

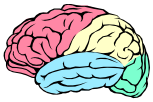
1. Create an array called rNum which contains max - min + 1 elements. The array elements are set to the values of each possible number that can be generated.
2. Beginning of a loop from i = 0 to i = num - 1 to generate each random number.
3. Generate a random number r between i and max - min inclusive. In this case r is a random index for the array rNum.
4. Add rNum(r) to the final array of random numbers.
5. Overwrite rNum(r) with the value in rNum(i). This ensures we don't re-use any numbers if the same r value was generated later on.
6. End of loop at step 2.

A possible algorithm using the above logic is reproduced in *Fig 4.23*:

```
BEGIN randArray(min, max, num)
    FOR i = 0 TO max - min
        rNum(i) = i + min
    NEXT i
    FOR i = 0 TO num - 1
        r = random integer from i to (max-min) inclusive
        arr(i) = rNum(r)
        rNum(r) = rNum(i)
    NEXT i
    RETURN arr
END randArray
```

Fig 4.23
Third attempt randArray algorithm.

In terms of performance analysis we have two loops but they are not nested. Essentially we separately iterate through two lists processing data, so roughly a $2n$ process. In terms of Big O, coefficients of n terms are not relevant so the algorithm is $O(n)$ – a vast improvement over the first two attempts.



Consider the following desk check:

A desk check of the third attempt randArray algorithm (*Fig 4.23*) using the call randArray(7, 10, 3). That is, generate three unique numbers from the numbers 7, 8, 9 and 10. We assume the random numbers generated for r during processing are 1, 3, 3

rNum				i	r	arr		
(0)	(1)	(2)	(3)			(0)	(1)	(2)
7				0				
	8			1				
		9		2				
			10	3				
	7			0	1	8		
			7	1	3		10	
			9	2	3			7



GROUP TASK Activity
Perform desk checks of each of the three algorithms developed in this section. Use calls to randArray with a small range and which generate a small number of random numbers to simplify the task.



GROUP TASK Discussion
There are certain sets of input parameters which are not valid. Explain how the input parameters could be validated to ensure the function operates appropriately.



GROUP TASK Practical Activity
Code the three versions of the randArray algorithm in a language of your choice. Devise a test subroutine to record the time taken for each version to generate large sets of unique random numbers. Do the results reflect our Big O calculations?



HSC style questions:

1. A particular language uses the RND function to generate a random floating point number from 0 up to but not including 1.

Which statement will generate a random floating point number from 20 up to but not including 36?

(A) Let R = (RND * 36) + 20

(B) Let R = (RND * 20) + 36

(C) Let R = (RND * 16) + 20

(D) Let R = (RND * 20) + 16

2. A language contains the function RAND, which randomly generates either 0 or 1. Which of the following expressions would best generate a random integer between 1 and 4 inclusive?

(A) 3*RAND + 1

(B) RAND + RAND + RAND + RAND

(C) RAND + RAND + RAND + 1

(D) 2*RAND + RAND + 1

Question 3.

A particular high-level programming language includes the function RINT. The RINT function generates a 1 byte random integer in the range 0 to 255 inclusive. RINT is the only random number function included within the language.

This language also includes the function INT(*x*). The INT(*x*) function returns the whole number part of *x* without rounding, for example INT(3.789) returns 3.

Sue is developing a game that will make extensive use of random integers. In particular the function needs to generate random numbers within the range of usual screen resolutions – up to a maximum of say 3000. She intends to use the function to generate random coordinates for placing characters on the screen.

Sue has written a function to generate a random integer within a given range – the algorithm for Sue's function is reproduced below.

```
1       BEGIN RanInt (Low, High)
2           tmpA = RINT
3           Range = High – Low + 1
4           tmpB = tmpA / 256 * Range
5           tmpC = INT(tmpB)
6           tmpD = tmpC + Low
7           RETURN tmpD
8       END
```

Sue has also created a routine which includes three inputs to test her RanInt function; example output is shown below. After running her test routine numerous times Sue forms the opinion that there is something wrong with her RanInt function. Not all numbers in the range are generated equally – indeed for large ranges many numbers are never generated. Three examples of the output from her test routine when the inputs are 1, 3, and 10000 are reproduced below:

```
Low: 1
High: 3
RanInts: 10000
1 was generated 3355 times.
2 was generated 3324 times.
3 was generated 3321 times.
```

```
Low: 1
High: 3
RanInts: 10000
1 was generated 3363 times.
2 was generated 3319 times.
3 was generated 3318 times.
```

```
Low: 1
High: 3
RanInts: 10000
1 was generated 3359 times.
2 was generated 3321 times.
3 was generated 3320 times.
```

(a) Perform a desk check of Sue's RanInt function if Low is 10, High is 20 and the RINT function returns 64.

(b) Construct an algorithm for Sue's test routine that generates output formatted in the same manner as the three examples shown in the question.

(c) Identify and describe problems with Sue's RanInt function that causes the unequal generation of random numbers and causes some numbers to never be generated.

Suggested Solutions

1. (C)

2. (D)

Question 2 explanation:

- Answer (A) will either generate 1 or 4 when RAND returns a 0 and 1 respectively. It is not possible for 2 or 3 to be generated.
- Answer (B) will generate 0 if all RAND calls return 0 and in addition not all numbers are generated equally (as in (C)).
- Answer (C) generates the correct range of numbers however they are not generated with equal probability. There is only one way to generate a 1 and a 4, namely 0+0+0+1 and 1+1+1+1 respectively. However there are three ways to generate a 2, namely 1+0+0+1, 0+1+0+1 and 0+0+1+1, similarly for the number 3.
- Answer (D) is correct as there is only one way to generate each of the numbers and hence they are equally likely. 2*0 + 0 + 1 = 1, 2*0 + 1 + 1 = 2, 2*1 + 0 + 1 = 3 and 2*1 + 1 + 1 = 4.

Question 3.

(a)

Low	High	tmpA	Range	tmpB	tmpC	tmpD	RanInt
10	20	64	11	2.75	2	12	12

(b)

```
BEGIN TestRanInt
    Get Low, High, NumInts
    Initialise Total(NumInts) to zero
    Count=1
    WHILE Count <= NumInts                          } Generate and tally
        Temp = RanInt(Low, High)                    } random integers.
        Increment Total(Temp)                       }
        Increment Count                             }
    ENDWHILE
    Display "Low: " Low
    Display "High: " High
    Display "RanInts: " NumInts
    Count = Low
    WHILE Count <= High                             } Display the
        Display Count " was generated " Total(Count) " times."  } tallied results.
        Increment Count                             }
    ENDWHILE
END TestRanInt
```

(c) The programming language's RINT function generates 256 distinct values. Each of these distinct values when processed by the algorithm returns a single value within the specified range. If the range does not divide equally into 256 then each number in the range cannot be generated equally. For example a range of 4 divides equally into 256, 256/4 = 64, hence all 4 numbers are generated equally. Consider the sample output in the question. This range of 3 does not work correctly as 256/3 = 85.333… This means that 1s are generated more often than 2s and 3s. On average a 1 is generated 86/256 of the time, whilst 2s and 3s are generated 85/256 of the time.

Furthermore any range containing more than 256 values cannot be generated by a single RINT call. For example if the range is 300 then 300-256=44 values. These 44 values can never be generated. The requirements require random integers with ranges up to 3000. For a range of 3000, 2744 values will never be generated.



GROUP TASK Discussion
Discuss strategies that could used to correct the problems present in Sue's RanInt algorithm.



GROUP TASK Activity
Write an algorithm that will fairly generate numbers up to 3000 using the languages RINT function.



GROUP TASK Activity
Write an algorithm that reads the screen resolution and then generates a random point on the screen using the RanInt function.

SET 4C

1. Which term describes the process of joining two strings to form a new string?
   (A) addition
   (B) concatenation
   (C) inserting
   (D) mixing

2. If RND generates a float from 0 up to but not including 1, which of the following generates floats in the range 4 up to but not including 10?
   (A) INT(RND * 7) + 4
   (B) INT(RND * 6) + 4
   (C) RND * 6 + 4
   (D) RND * 7 + 4

3. Which of the following is the fairest way to generate the sum of rolling 2 dice using the *Fig 4.23* randArray algorithm
   (A)
   ```
   arr= randArray(1, 6, 2)
   sum = arr(0) + arr(1)
   ```
   (B)
   ```
   sum = randArray(1, 6, 1) * 2
   ```
   (C)
   ```
   sum = randArray(1, 6, 2) +
   randArray(1, 6, 2)
   ```
   (D)
   ```
   sum = randArray(2, 12, 1)
   ```

4. Using the algorithms in *Fig 4.17* to *4.20*, what string is returned by the following
   ```
   Delete(6,10,Insert(6,"Dead ","Drop Fred"))
   ```
   (A) Drop Dead Fred
   (B) Drop Fred Dead
   (C) Drop Dead
   (D) Drop Fred

5. Making a copy of a section of a string is called:
   (A) inserting.
   (B) deleting.
   (C) extracting.
   (D) concatenation.

6. A parameter can be best described as:
   (A) a value set by one routine that can be accessed by other routine s.
   (B) a value or variable that provides the input to another routine.
   (C) a variable that provides a location for the output from a routine.
   (D) A value or variable that is sent and returned from a routine.

7. The original string will be smaller after:
   (A) inserting.
   (B) deleting.
   (C) extracting.
   (D) concatenation.

8. Joining 2 or more strings involves the process of:
   (A) inserting.
   (B) deleting.
   (C) extracting.
   (D) concatenation.

9. Using the *Fig 4.23* randArray algorithm, which call would be best to simulate shuffling a standard pack of 52 cards?
   (A) arr= randArray(1, 13, 52)
   (B) arr= randArray(0, 51, 51)
   (C) arr= randArray(1, 52, 1)
   (D) arr= randArray(1, 52, 52)

10. Using the algorithms in *Fig 4.17* to *4.20* and if str="The cat sat on the mat", which string is returned by the following
    ```
    Insert(5,"dog ",Delete(5,11,Extract(5,18,str)))
    ```
    (A) cat dog the
    (B) catdogthe
    (C) sat on the mat
    (D) The dog

11. 'Extracting, deleting and inserting can be likened to the copy, cut and paste commands in many software applications'. Explain this statement. Use examples to justify your explanation.

12. Write an algorithm which simulates tossing a coin 1,000,000 times and then reports the percentage of heads and the percentage of tails.

13. Functions to find and replace substrings are often used in text editors including word processors. Write an algorithm which will search and replace each occurrence of a string with another string. Your algorithm may call the algorithms in *Fig 4.17* to *4.20*.

14. Write an algorithm which repeatedly draws 5 cards from a complete standard pack until the cards include all 4 aces. Finally, display the number of draws required to draw all 4 aces.

15. An array contains dates where each date is represented as a string in the form DD/MM/YYYY. Write an algorithm which will check each date in the array is legitimate. This includes legal number of days, months and years. For example, 31/01/2012 is legal, but 31/09/2012 is not as there are only 30 days in September. Also 29/02/2012 is legal as it is a leap year, whereas 29/02/2013 is not a legitimate day as 2013 is not a leap year.

CUSTOM-DESIGNED LOGIC USED IN SOFTWARE SOLUTIONS

Before commencing work on an algorithm, it is necessary to know the inputs and the desired outputs, together with an idea of the processing required to transform the inputs into the outputs. This information is often obtained from models of the system, in particular dataflow diagrams, structure charts and IPO diagrams. Once the inputs, processing and outputs are determined, work commences on designing data structures and assigning the appropriate data types. This information is documented using a data dictionary. Finally, an algorithm that describes the logic of each subroutine within the solution can be developed.

DATA STRUCTURES AND FILES

Efficient and appropriate data structures greatly assist in the subsequent development of an algorithm. In this section, we examine some common data structures - multi-dimensional arrays, and arrays of records. We also examine sequential and random access files. There are numerous other data structures available to programmers such as linked lists, stacks, queues, binary trees and hash tables. If you study computer science at tertiary level no doubt you will encounter all these data structures and more.

Multi-Dimensional Arrays



Index
An integer value used to denote a particular data item held in an array. Also called a dimension or subscript.

In the Preliminary course, we examined single-dimensional arrays. Remember that all items in an array must be of the same data type. An example of this would be an array that is used to store the different names of people. That is: `Name(0)`, `Name(1)`, `Name(2)`…. Each of these items must hold the same type of data, and in this example the type would be a string. Multi-dimensional arrays are also of one data type. Rather than one dimension or index, multi-dimensional arrays can have a number of indexes.

Often the dimensions or indexes for an array are called subscripts, because of their similarity to subscripts commonly used in mathematics. The dimensions of an array are usually given a name that describes their purpose. An example of this would be an array that is used to store the assessment task results of your class.

```
AssessResults(StudentNum, TaskNum)
```

Let us examine some possible data that could be held in this array. `AssessResults(11,4)=72`. This means that student number 11 on the 4th assessment task scored 72 marks. The names given to the indexes are only for documentation purposes. The following code fragments are functionally identical:

```
Table = 5
Chair = 4
AssessResult(Table,Chair) = 85
```

```
Dog = 5
Cat = 4
AssessResult(Dog,Cat) = 85
```

Two-dimensional arrays can be visualised as tables where each cell in the table holds data of the same type. Usually the indexes are used as a way of grouping the data. In the AssessResults array, StudentNum is the first index and groups the data according to individual students. TaskNum, the second index or dimension, groups the data by the task number.

		Assessment Task Number				
		1	**2**	**3**	**4**	**5**
Student Number	**1**	60	65	67	68	69
	2	65	73	70	62	71
	3	72	60	58	71	69
	4	85	86	80	82	66
	5	68	74	85	90	88
	6	58	55	52	59	62
	7	70	75	70	71	69

Fig 4.24
Two-dimensional arrays can be represented as tables.

Three-dimensional arrays can be thought of as a cube or a rectangular prism of data where each block contains a data item. Let us extend the AssessResults example so we can store data about all the classes in Year 12.

AssessResults(ClassNum, StudentNum, TaskNum)

If your SDD class is class number 6, your student number is 14, and you have just completed the 2nd assessment task then AssessResults(6,14,2) would be used to store your result in this assessment task.

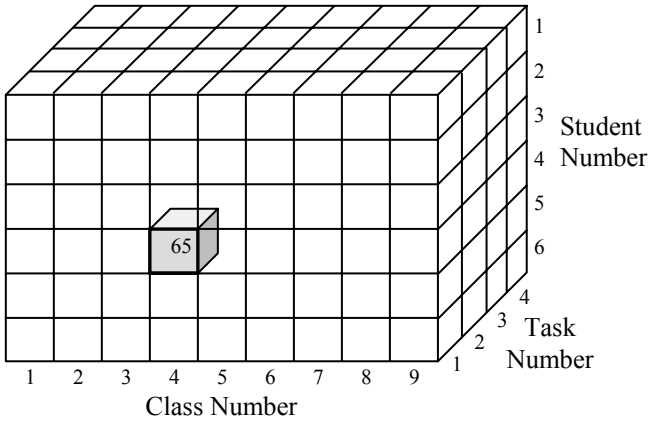


Fig 4.25
Three-dimensional arrays can be represented as rectangular prisms.
In the above diagram AssessResults(4,4,1) holds a value of 65.

Notice that each block is determined by a unique combination of ClassNum, StudentNum and TaskNum, and that the order of the indexes is very important. A particular ClassNum allows access to all the data about that class. For example, if ClassNum is held constant at 6, then altering the student and task numbers will access all the assessment results for Class 6. Similarly, holding a StudentNum constant provides all the assessment results for that particular student.



GROUP TASK Activity
Examine *Fig 4.25*. Identify the blocks that hold (a) all the assessment marks for a single student, (b) all the marks for a particular assessment task, (c) a student's assessment marks in one of their classes.

Let us now extend our array to 4 dimensions. In our example the 4th dimension will represent the year level, which can take values from 7 through to 12.

AssessResults(YearLevel, ClassNum, StudentNum, TaskNum)

Remember, each item stored in this array is a mark from 0 to 100: that is, each data item is of the same type. An example data item could be:

AssessResults(9, 4, 16, 2) = 68

This element of the array contains a mark of 68 obtained in the 2nd task by student number 16 who is in the 4th year 9 class. Note that in this structure there can be a student number 16 in each of Years 7, 8, 9, 10, 11 and 12. The important concept is that each combination of indexes *must* be unique.

Let us now look at some algorithms that use our AssessResults array. We will assume there is a maximum of thirty students in a class and that a mark of -1 denotes a result not to be included.

Following is an algorithm to find the class average for a particular task.

```
BEGIN ClassTaskAverage
    Get YearLevel,ClassNum,TaskNum
    StudentNum=1
    Total=0
    NumStudents=0
    WHILE StudentNum <=30
        IF AssessResults(YearLevel,ClassNum,StudentNum,TaskNum) >=0 THEN
            Add 1 to NumStudents
            Total=Total+AssessResults(YearLevel,ClassNum,StudentNum,TaskNum)
        ENDIF
        Add 1 to StudentNum
    ENDWHILE
    IF NumStudents > 0 THEN
        Average=Total/NumStudents
        Display Average
    ENDIF
END ClassTaskAverage
```

Fig 4.26

Algorithm to calculate the class average for a particular assessment task.

Notice that a particular task for a particular class is determined by the three indexes YearLevel, ClassNum and TaskNum. These values are obtained whilst the StudentNum is variable. Varying the student number index StudentNum allows access to all the marks achieved on this particular task.



GROUP TASK Discussion

The above algorithm ignores array elements that contain -1. How is this achieved?



GROUP TASK Discussion

It is possible for more than one class to have the same class number. Explain why this is reasonable. Use examples to explain your answer.

The algorithm that follows finds the average on all tasks completed by a particular year level, assuming a maximum of 100 classes per year level, 30 students per class and 10 tasks per class.

```
BEGIN YearLevelAvg
    Get YearLevel
    Total=0
    NumStudents=0
    ClassNum=1
    WHILE ClassNum<=100
        StudentNum=1
        WHILE StudentNum<=30
            TaskNum=1
            WHILE TaskNum<=10
                IF AssessResults(YearLevel,ClassNum,TaskNum,StudentNum)>=0
                    Add 1 to NumStudents
                    Add AssessResults(YearLevel,ClassNum,TaskNum,StudentNum) to Total
                ENDIF
                Add 1 to TaskNum
            ENDWHILE
            Add 1 to StudentNum
        ENDWHILE
        Add 1 to ClassNum
    ENDWHILE
    IF NumStudents >=0 THEN
        Average=Total/NumStudents
        Display Average
    ENDIF
END YearLevelAvg
```

Fig 4.27
Algorithm to find the average mark for all results for a particular year level.



GROUP TASK Discussion
There are three loops in the above algorithm. What is the purpose of these loops?



GROUP TASK Discussion
Array elements that do not contain marks have a value of -1. Design an algorithm that will initialise all the array elements to -1.

Arrays of Records

In the Preliminary course, we examined records and determined that a record is a data structure that contains a number of fields. Each of the fields contained in a record can be of a different data type. Once a record is declared, it becomes a data type in the same way as any other predefined, simple data type.

For example, a record used to store Name, Address and Date of Birth details could have the following fields:

Field Name	Data Type
Surname	String
Cname	String
Title	String
Street	String
Suburb	String
Postcode	String
DOBDay	Integer
DOBMonth	Integer
DOBYear	Integer

If a record called MyDetails is created then data could be assigned as follows:

```
MyDetails.Surname = "Nerk"
MyDetails.Cname  = "Fred"
MyDetails.Title = "Mr"
MyDetails.Street = "12 Foogle Road"
MyDetails.Suburb = "Fendalton"
MyDetails.Postcode = "1234"
MyDetails.DOBDay = 25
MyDetails.DOBMonth = 6
MyDetails.DOBYear = 1988
```

If a second record of the same type called OtherDetails was created, and it was also to contain all of Fred Nerk's details, then the following assignment statement would achieve this:

```
OtherDetails = MyDetails
```

Records can be treated as single data types, or each of the fields contained within a record may be accessed independently. Let us use an example of the creation of an address book that contains the details of 100 people. Each record could be created individually, however the process would be tedious and is unnecessary. It would be far more efficient to create an array of these records instead. The array could be called MyContacts. If the 40th person's address details needed to be accessed, then the following items would be examined:

```
MyContacts(40).Street
MyContacts(40).Suburb
MyContacts(40).Postcode
```

If the 3rd contact needed to be swapped with the 4th contact, then the following would achieve this:

```
Temp = MyContacts(3)
MyContacts(3) = MyContacts(4)
MyContacts(4) = Temp
```

An array of records can be likened to a table (or relation) in a database. If the MyContacts array of records were viewed as a table, it would look like the following:

Index	Surname	CName	Title	Street	Suburb	Postcode	DOBDay	DOBMonth	DOBYear
1	Davis	Sam	Mr	4 Data St	Romany	1234	14	5	1988
2	Hart	Rob	Mr	1 Cap Ln	Billings	9876	29	10	1951
3	Jackson	Jack	Mr	7a Pot Rd	Potfield	8765	13	9	1983
4	Unders	Wilma	Mrs	18 May St	Kingsley	1005	30	1	1977
5	Milson	Margaret	Miss	17 Jax Cr	Watson	5633	23	6	1966



Consider the following:

An address book such as the one described above is being created in Visual Basic. The following fragment of code is used to declare the array of records MyContacts:

```
Structure Address
   Surname As String * 30
   Cname As String * 30
   Title As String * 4
   Street As String * 40
   Suburb As String * 30
   Postcode As String * 4
   DOBDay As Integer
   DOBMonth As Integer
   DOBYear As Integer
End Structure

Dim MyContacts(100) As Address
```

Firstly, a new data type called Address is defined using the Structure statement. An array called MyContacts is then declared with index values from 0-100 and of data type Address.

We noted earlier that copies of records can be made by simply assigning one record to another. For example, MyContacts(20) = MyContacts(21) will copy all the data held in each field of MyContacts(21) into MyContacts(20) – in effect a new record is created which overwrites the old MyContacts(21). If we edit MyContacts(21).Surname it has no effect on the value in MyContacts(20).Surname.

This is not what occurs when dealing with arrays. For example, MyArr = MyContacts does not make a copy of all the records in MyContacts rather it just causes the identifier MyArr to point to the same array as the MyContacts identifier. There is only one array of records which can now be accessed using either the MyArr or MyContacts identifier. Altering MyContacts(20).Surname will also change MyArr(20).Surname because they refer to the same memory location**.**



GROUP TASK Research
Examine at least two other programming languages and create declaration statements that achieve the same affect as the Visual Basic code above.



GROUP TASK Activity
Declare the above array of records in a programming environment with which you are familiar. Create and implement an algorithm to enter data into this structure. Try copying the complete array and copying individual records to confirm the behaviour is as described above.

An array of records can also be multi-dimensional in the same way as a simple data type array. In addition, records may also contain fields that themselves are arrays.



Consider the following:

A data structure is required to store qualifying data on cars entered for an upcoming motor racing event. Some sample data is given below:

Number	Model	Driver	Lap_1	Lap_2	Lap_3	Lap_4	Lap_5
25	Holden	Davis	185.2	180.3	181.1	179.9	179.7
05	Holden	Walker	183.3	181.7	179.5	184.9	180.2
17	Ford	Harrison	179.4	181.8	183.5	178.6	182.7

An individual record could contain the following fields:

```
Number        String
Model         String
Driver        String
Lap(LapNum)   Array of real numbers
```

An array of records of this data type would be used to store the qualifying details for all the cars entered. The array could be called Qualifying. Let us examine the contents of some items in this array of records:

```
Qualifying(1).Number = "25"
Qualifying(1).Driver = "Davis"
Qualifying(1).Lap(3) = 181.1
Qualifying(2).Lap(5) = 180.2
```

The data structure chosen for a particular problem must be designed with consideration given to the processing required in the algorithm. Selecting an appropriate data structure will often greatly reduce the complexity of the final algorithm. In the above Qualifying data structure, calculating the fastest lap time for each car is a much simpler task because the lap times are in an array which is part of each car's record.

```
BEGIN CalculateFastestLaps
    Count = 1
    WHILE Count <= Total cars
        Num = 2
        Fastest = 1
        WHILE Num <= 5
            IF Qualifying(Count).Lap(Num) < Qualifying(Count).Lap(Fastest) THEN
                Fastest = Num
            ENDIF
            Num = Num + 1
        ENDWHILE
        Display Qualifying(Count).Driver, Qualifying(Count).Lap(Fastest)
        Count = Count + 1
    ENDWHILE
END CalculateFastestLaps
```

Fig 4.28
Algorithm to calculate and display the fastest lap time for each car.



GROUP TASK Discussion
Describe in words the operation of the above algorithm. In particular describe how the lap times are accessed from the Qualifying array.



GROUP TASK Activity
Desk check the above algorithm using the sample data in the associated table.

Files

A file, in terms of software development, means a collection of data that is stored in a logical manner. Normally, files are stored on secondary storage devices, usually a hard disk that is separate from the application program itself. There are essentially two methods of storing and accessing data in a file:

- Sequential files
- Relative (or random) files

Sequential Files

Data is stored in sequential files in a continuous stream. The data must be accessed from beginning to end. An audiocassette is a sequential storage medium for audio data. To play the third track on an audiocassette requires fast forwarding through tracks one and two. Sequential files operate in the same manner. To access data stored in the middle of a sequential file requires reading all the preceding data.

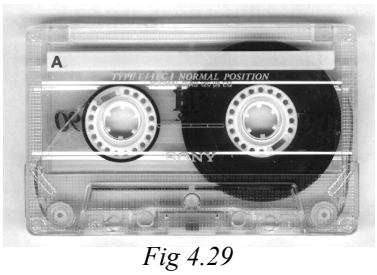


Fig 4.29
Audiocassettes store music tracks sequentially.

The data stored in a sequential file may have some structure, however the structure is not stored as part of the file. Applications that access sequential files must know about the structure of the file. Text files are sequential files; the data within a text file is merely a collection of ASCII or Unicode characters.



Sentinel value
A dummy value used to indicate the end of data within a file. Sentinel is from the word sentry, a sentry being a guard who prevents passage of unauthorised persons.

When using sequential files it is necessary to structure the data yourself. Sentinel values can be used to indicate logical breaks in the data and also to indicate the end of the file. For example, tab characters may be used between fields and carriage return characters between records. Often a particular string, such as "ZZZ", may be used as a sentinel to indicate the end of a sequential file. When a sentinel value is used to signal the end of a file care is required to ensure the sentinel value is not processed as if it were data. When reading sequential files it is common to use an initial or priming read before the main processing loop to detect files which only contain the sentinel. *Fig 4.30* describes the logic used to read files that include a final sentinel value.

```
BEGIN ReadFile
   Open file for input
   Read item from file
   WHILE item ≠ "ZZZ"
      Process item
      Read item from file
   ENDWHILE
   Close file
END ReadFile
```

Fig 4.30
Generic algorithm to read a sequential file with sentinel.

Many programming languages include their own commands for writing to and reading from sequential files. These commands operate well when the file will only be accessed using the inbuilt commands. For instance the language's write command is used to create files and the language's read command is used to get data from the file. Essentially the format of the text file is defined and determined by the language's commands and the programmer is relieved of much of the detailed work.

Sequential files can be opened in one of three modes – input, output and append. Input is used to read the data within the file, output is used to write data to a new file and append is used to write data commencing at the end of an existing file. Notice it is generally not possible to commence writing from the middle of an existing sequential file. This is because the length of data items within the file is unknown and hence there is no way to ensure required data is not overwritten. Relative (or random access) files overcome this restriction.



Consider the following:

A sequential file called animal.txt contains a list of animal names separated by commas and ending with the sentinel value "ZZZ". To access a particular animal name requires reading each preceding animal name. A particular animal name cannot be directly accessed.

A possible animal name file follows:

Ant,Hen,Monkey,Pig,Rat,Cow,Tiger,Warthog,Dog,Elephant,Frog,Goat,ZZZ

If we wish to read this file into an array called Animals, we need to first open the file, then read and examine the value of each character one at a time. Commas tell us when we have reached the end of a word, and ZZZ is the sentinel value that indicates the end of the file. Once processing is complete we close the file.

```
BEGIN ReadSequentialFile
   Open animals.txt for input
   Set Index to 1
   WHILE Animals(Index) does not equal "ZZZ"
      Read Character from animals.txt
      IF Character is a comma THEN
         Add 1 to Index
      ENDIF
      Animals(Index) = Animals(Index) + Character
   ENDWHILE
   Close animals.txt
END ReadSequentialFile
```

Fig 4.31
Algorithm to read the animal name sequential file.



GROUP TASK Activity
There is a deliberate error in the above algorithm. Perform a desk check of the algorithm using the sample file and determine the error. Suggest a possible way of correcting this error.



GROUP TASK Activity
The above algorithm reads the sentinel value "ZZZ" into the array Animals. Redesign the algorithm so that the sentinel value is not read into the array.



GROUP TASK Discussion
The above file includes the sentinel value "ZZZ". This will be a problem if extra animals are to be appended to the file. Discuss the issue.



GROUP TASK Activity
Many languages use a separator (such as the commas in the above animal file) so that each data item can be read directly into a variable. Rewrite the algorithm in *Fig 4.31* so it reads each animal name directly into the array.

Relative (or random access) files

Relative refers to the fact that there is a known structure to these files which allows the start of individual records to be determined. As each record within the file is exactly the same length then the relative position of each record within the file can be used to access individual records. In effect the position of each record in the file is used as a key to allow direct access to each record. For example, if records are 30 bytes long then the 100th record will commence after the 3000th byte in the file.

Random access refers to the ability to access records in any order. Unlike sequential files, it is possible to read the 10th record, then the 2nd record, then the 50th record and so on in any desired random order. With sequential files we must read from the start of the file through to the end of the file, hence if we require the 100th record we must first read each of the preceding 99 records. Furthermore when writing data we are able to edit or update individual records within a file, which was not possible using sequential files.



Fig 4.32
Audio compact discs provide random access to individual tracks.

Random access or relative files can be likened to audio compact discs. With a music CD, individual tracks can be played in any order. Many CD players include a random function whereby tracks are played in a random order. The structure of an audio CD is stored on the CD. If you wish to play track five, the laser head jumps directly to the start of track five. Random access files allow individual data items to be accessed directly without the need to read any of the preceding data. In fact, random access files are often known as direct access files.

Relative files are used to store records. Each record is the same data type and must be of precisely the same length. Complete records are read and written to relative files. The programming language is able to able to determine the precise byte where a record begins because all records are the identical length. Individual fields within records can be identified precisely because the exact length and structure of each record is known. Fields containing strings that are of differing lengths are usually padded out using the blank character (ASCII code 32).



Consider the following:

A random access file is used to store the details of products sold by a particular business. The size of the fields within each record, are described in *Fig 4.33*. The number of characters in the four string fields total 18 + 30 + 5 + 5 = 58 characters. Each character requires one byte of storage space, plus another 2 bytes to store the cost, giving a total of 60 bytes. Every record in this file is therefore 60 bytes long.

Name	String of length 18
Description	String of length 30
SupplierID	String of length 5
ProductID	String of length 5
Cost	16 bit Integer

Fig 4.33
Record definition for the products file.

Suppose we wish to access the Description of the ninth record. The system calculates the byte number where this data is stored: eight records times 60 bytes gives 480 bytes, that is, record number 9 begins in position 481. These calculations are transparent from the point of view of the programmer. The programmer refers to the record by its index, nine in this case. Once the record is read into memory, the

description field can be accessed. Most high-level languages require retrieval of complete records. In high-level languages the keyword Get is commonly used to retrieve records, and Put is used to store records. When writing algorithms it is common to use Read and Write when dealing with both sequential and relative access files.



GROUP TASK Investigation
Examine the documentation of a programming language with which you are familiar. Create a list of commands available for accessing relative files. Describe the function of each command you find.



GROUP TASK Investigation
The use of relative files is becoming uncommon due largely to the widespread use of XML and DBMSs. Research and discuss why this is the case.



HSC style questions:

1. In the following line, "=" means "is assigned", whilst "= =" means "equals".

```
found = myArray(index) = = itemToFind
```

In regard to the data type of each identifier used in the above line, which row of the table is most likely to be correct?

	found	index	myArray	itemToFind
(A)	Boolean	Integer	Array of Strings	Integer
(B)	Integer	Real	Array of Integers	Integers
(C)	Integer	Integer	Array of Integers	Integer
(D)	Boolean	Integer	Array of Strings	String

2. Which of the following best describes the expression Animal(34).Species?

(A) The Species field of the 34th record in the Animal array.

(B) The Species field of the record with index 34 in the Animal array of records.

(C) The 34th Species record in the Animal array.

(D) The 34th Species field in the Animal record.

3. The rainfall during each month over a twenty year period is to be read from a file and processed to determine the average annual rainfall and the average rainfall for each month. What is the most appropriate data structure?

(A) An array of records

(B) An array with two dimensions

(C) An array with one dimension

(D) A series of records

Question 4.

The popular word game Target requires you to see how many words of four or more letters you can make from the letters in the grid. Each word must include the centre letter. Thus, in the grid at right the words CIDER, GRIND, RECORDING are all valid.

G	N	R
R	**D**	C
I	E	O

Each letter in the grid must be used only once in a word. If a letter appears twice (such as the R in the grid at right) then there may be 2 R's in the word. Thus **R**ECO**R**DING is a valid word.

You are developing Target into an on-line game. The software will generate a random grid based on a 9 letter word randomly selected from all possible 9 letter words. During a game the player enters one word at a time. If the word is valid then the player is awarded one point. The game ends when the player enters a blank.

There are two relative files available that have been created from an electronic dictionary. The first file (w49.txt) contains a total of 61419 words, which are all the words from the dictionary containing from 4 to 9 letters. The second relative file (w9.txt) contains 13062 words, which is all the 9 letter words from the dictionary. The words in both files are in ascending order.

The following algorithms have already been created:

```
BEGIN Target
    GenerateGrid
    Score = 0
    Input Word
    WHILE Word is NOT blank
        IF CheckWord(Word) THEN
            Increment Score
        ENDIF
        Display Score
        Input Word
    ENDWHILE
END Target
```

```
BEGIN GenerateGrid
    ReadGridWord
    ShuffleLetters
    DisplayGrid
END GenerateGrid
```

```
BEGIN CheckWord(Word)
    Check = False
    IF LengthOK(Word) THEN
        IF LettersInGrid(Word) THEN
            IF InDictionary(Word) THEN
                Check = True
            ENDIF
        ENDIF
    ENDIF
    RETURN Check
END CheckWord
```

(a) Construct a structure diagram based on the above algorithms.

(b) Design an algorithm for the ReadGridWord routine that loads the array Grid() with a random word from the relative file called w9.txt. For example, if the random word found in the file is 'RECORDING' then the routine ends with Grid(0)='R', Grid(1) ='E', Grid(2)='C',... Grid(8)='G'.

(c) Two versions of the ShuffleLetters routine have been written by Joe and Jane.

Joe's Version:

```
BEGIN ShuffleLetters
    FOR i = 8 TO 1 STEP -1
        Ri = INT(RAND * i)
        Swap(Grid(i), Grid(Ri))
    ENDFOR
END ShuffleLetters
```

Jane's Version:

```
BEGIN ShuffleLetters
    FOR i = 0 TO 8
        Ri = INT(RAND * 9)
        Swap(Grid(i), Grid(Ri))
    ENDFOR
END ShuffleLetters
```

Describe suitable testing that could be performed to compare the performance and randomness of these two solutions.

(d) The following LettersInGrid routine checks all letters in the entered word are present in the Grid array according to the rules of the game.

```
1.  BEGIN LettersInGrid(Word)
2.      FOR i = 0 TO 8
3.          Used(i) = False
4.      ENDFOR
5.      AllOk = True
6.      WordLength = Number of chars in Word
7.      c = 1
8.      WHILE c <= WordLength AND AllOk
9.          i = 0
10.         Found = False
11.         WHILE i <= 8 AND NOT(Found)
12.             IF Grid(i) = letter in position c of Word AND Used(i) = False THEN
13.                 IF Grid(i) = Grid(4) AND Used(4) = False THEN
14.                     Used(4) = True
15.                 ELSE
16.                     Used(i) = True
17.                 ENDIF
18.                 Found = True
19.             ENDIF
20.             i = i + 1
21.         ENDWHILE
22.         IF NOT(Found) THEN
23.             AllOk = False
24.         ENDIF
25.         c = c + 1
26.     ENDWHILE
27.     IF AllOk AND Used(4) THEN
28.         LettersInGrid returns True
29.     ELSE
30.         LettersInGrid returns False
31.     ENDIF
32. END LettersInGrid
```

Explain how each of the following Boolean variables is used by the LettersInGrid algorithm:

- Used(i)
- AllOk
- Found

(e) Provide the algorithm for the InDictionary routine. This routine is to use a binary search strategy to check if the word entered by the player is in the w49.txt file.

Suggested Solutions

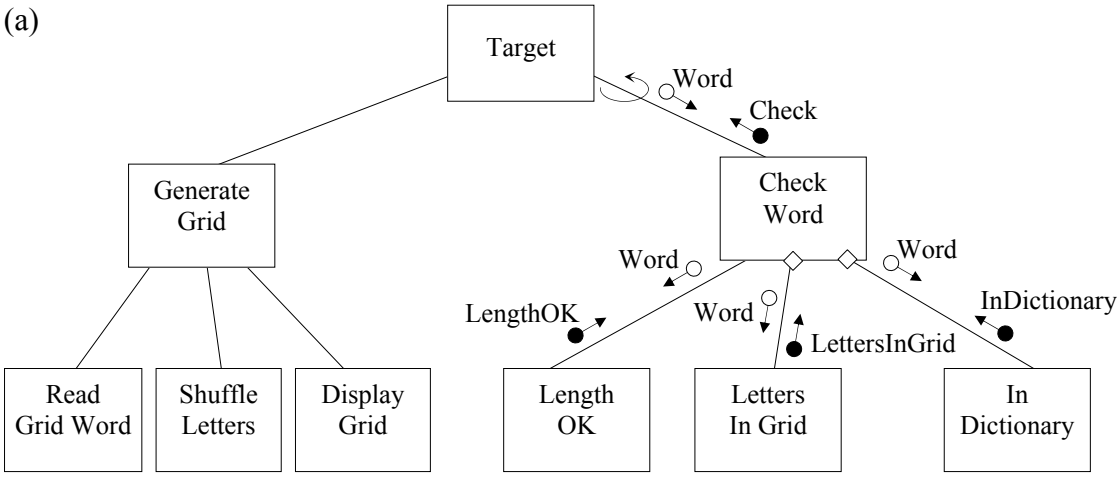
1. (D) Found is Boolean so (A) and (D) are possible answers. Both myArray and itemToFind must be the same data type therefore (D) is the correct answer.

2. (B) The index is 34, which means the 35th array element if the array is indexed from zero. Therefore (B) is a better answer than (A).

3. (B) The rainfall readings are all of the same type so an array is appropriate. A 2 dimensional array would work best with one index from 0 to 19 for each of the 20 years and the other index from 0 to 11 for each month of each year.

Question 4.

(a)



(b)
```
BEGIN ReadGridWord
    r = Random integer from 1 to 13062
    Open w9.txt for relative access
    Read rth record in w9.txt into AllLetters
    FOR i =1 to length of AllLetters
        Extract ith character of AllLetters into Grid(i)
    END FOR
    Close w9.txt
END ReadGridWord
```

(c) To see the differences between the two approaches clearly, it would be necessary to write a driver that calls the routine say 10000 times. The driver would need to store the value of the timer just before the routine was called for the first time and then again after the completed number of iterations is complete. The difference between these two times gives the time taken by the routine, which can then be divided by the number of iterations. It is an easy matter to compare the time taken for the one approach, compared to the other.

To determine the 'randomness' of each approach, it is necessary to shuffle the same set of letters many times (millions of times) and see if each different combination of the letters is generated approximately the same number of times. There are a finite number of different combinations possible (9 factorial), it would be useful to use a second (large) array which keeps count of the number of times that each different combination occurs. Sorting the array would then highlight any combinations that occur less or more than expected. The algorithm that generates an equal spread across all combinations would be the best in terms of generating a random combination of letters.

(d) Used(i) checks whether the i^{th} letter of the grid has already been 'ticked off' – that is, whether the i^{th} letter of the grid matches one of the letters in the word entered. This approach also takes into account letters that appear more than once in the grid – each of them must be 'ticked off' as being present in the entered word for the word to be valid. The relevant lines in the algorithm are lines 11 – 21.

AllOk is a flag that is set to indicate if any letter in the entered word is not found in the grid. This flag is set in lines 22 – 24, and causes the outside to loop to immediately exit. It is tested again after the loops in lines 27 – 31 to see if the

word is valid (and the centre letter was used) and then finally return the appropriate value.

Found is a flag that is set as soon as a letter in the entered word is found to match an unused letter in the grid (line 18). Found is the flag that is checked after every letter in the grid has been checked against the next letter in the word. Found is the flag that causes the AllOk flag to be set after every letter in the grid has been checked against the next letter in the word.

(e)
```
BEGIN InDictionary(Word)
    start = 1
    finish = 61419
    found = False
    Open w49.txt for relative access
    REPEAT
        mid-pt = INT((start + finish)/2)
        Read mid-ptth record in w49.txt into DictionaryWord
        IF DictionaryWord = Word THEN
            found = True
        ELSE
            IF DictionaryWord > Word THEN
                finish = mid-pt - 1
            ELSE
                start = mid-pt + 1
            END IF
        END IF
    UNTIL start >= finish OR found
    RETURN Found
    Close w49.txt
END InDictionary
```



GROUP TASK Discussion
Discuss changes that would need to be made to the InDictionary subroutine in (e) if sequential access was used rather than relative access.



GROUP TASK Discussion
In part (c) discuss in theoretical terms which of the two algorithms should perform the best and which is the fairest in terms of randomness.



GROUP TASK Practical Activity
Work in groups to implement the Target game. Cooperate to assign subroutines to members of the group. Thoroughly test the application.

SET 4D

1. Multi-dimensional arrays are useful for storing:
   (A) a number of data items of different types that need to be accessed together.
   (B) data of the same type that possesses a number of characteristics by which it can be grouped.
   (C) X and Y style coordinates that can later be graphed.
   (D) documentation information that will later assist with upgrades and program maintenance.
2. A value used to indicate the end of a data stream is called:
   (A) a sentinel value.
   (B) an end of file (EOF) character.
   (C) a flag.
   (D) a driver.
3. In the expression Products(Count), Count would most likely be:
   (A) a variable of type string.
   (B) the index for the array.
   (C) a variable of type integer.
   (D) an array of strings.
4. The ability to retrieve a particular record from a file is a feature of:
   (A) text files.
   (B) sequential files.
   (C) random access files.
   (D) binary files.
5. The names given to the indexes in a multi-dimensional array:
   (A) are used to determine the data type of the array.
   (B) can also be called dimensions or subscripts.
   (C) define the names of the variables that must be used to access data items held in the array.
   (D) are used for documentation purposes.
6. Fields within a record:
   (A) must hold data of the same type.
   (B) can be arrays.
   (C) can themselves be records.
   (D) All of the above.
7. Before work can commence on the development of algorithms it is vital to:
   (A) make sure the system model is logically correct in every detail. It must explain every data structure the algorithm will require.
   (B) know the details of the original problem in its entirety.
   (C) examine the hardware on which the final product will operate.
   (D) know the inputs and outputs, have an idea of the processing required, and have designed the data structures.
8. A sequential file uses a comma to separate fields. This is a poor choice of separator when:
   (A) the length of each data item varies.
   (B) the data itself may contain commas.
   (C) numeric data is stored in the file.
   (D) each data item is the same length.
9. An array of records:
   (A) can only have one index.
   (B) contains records that are all of the same type.
   (C) can only be accessed sequentially.
   (D) is used to store a single list of data of a particular type.
10. The expression Products(5).Price can be best described as accessing:
    (A) the fifth record in the array Products.
    (B) the field called Price in the Products record.
    (C) the fifth Price in the array called Products.
    (D) the Price of the record with array index 5 within the array Products.
11. If A=5, B=3, C=1 and D=6, what can you say about the data items accessed by each of the following expressions? Item(A,B,2), Item(D-C,3,A-B), and Item(2B-C,3C,D/B).
12. A teacher's class roll is used to store the personal details of each student in the class together with absence data on each student for each school day. Design a data structure that could be used to hold this information.
13. The class roll data from question 12 needs to be stored in a file. Would you use a sequential or a random access file for this purpose? Explain the advantages and disadvantages of each method of file storage in regard to this scenario.
14. Your local council is collecting statistics on the number of vehicles using your street each day. The data is collected using a device that automatically increments when a car passes. The council wishes to analyse the data in terms of the total traffic per day, per week and per month over a 12-month period. Design a data structure to store this data in a way that allows for easy calculation of the required totals.
15. Using your data structure developed in question 14, design an algorithm that will output, in table form, the daily, weekly and monthly traffic flow.

CREATING CUSTOM-DESIGNED ALGORITHMS

Once a model of the system has been created and data structures have been designed, the work of creating the algorithms can commence. Many solutions will involve the use of existing, standard modules such as the searches and sorts examined earlier in this chapter. Often the most difficult task for software developers is the creation of custom algorithms. It is difficult to learn how to write custom algorithms, as every problem presents new difficulties and challenges. Often many attempts will be necessary before a successful algorithm is developed. Some consolation can be had in the knowledge that the more algorithms you write, the easier it will become.

Let us now go through the process of creating some custom algorithms:

The card game 'snap'

In this section, we will create a series of algorithms to simulate the card game snap. Snap utilises a standard pack of playing cards. There are 4 suits: hearts, diamonds, spades and clubs. Each suit contains 13 cards: Ace, 2, 3, 4…10, Jack, Queen and King – a total of 52 cards. For this simulation, the game is played with two players. Each player in turn takes a card from their half of the pack and places it on the stack in the middle of the table. If the card placed has the same value as the previous card, the first player to put their hand on the stack wins the stack. The stack of cards is placed under the player's existing cards and the game continues. The winner is the player who eventually wins all the cards.

The following structure chart models this game:

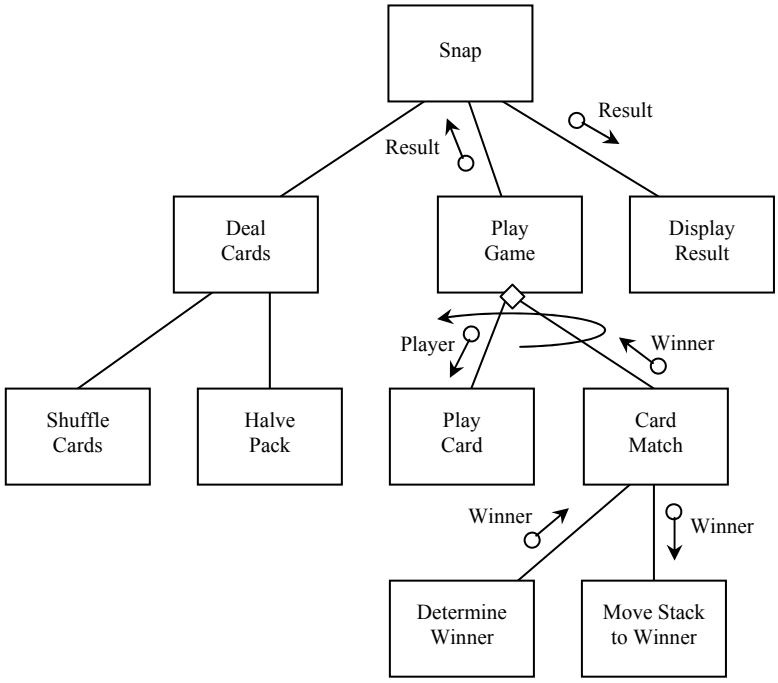


Fig 4.34
Structure chart for the 'snap' simulation.

Data structures for the card game 'snap'

We now need to think about and develop the data structures to be used. There are a total of 52 cards and each card will always be in one of three places throughout the game: either with one of the two players or in the centre stack.

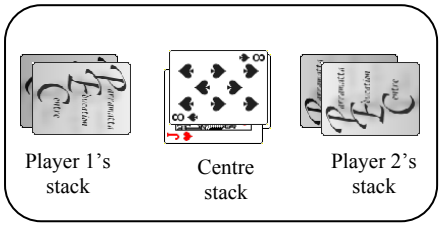


Fig 4.35
Three stacks that together contain all 52 cards in the pack.

Initial thoughts are that we could create three separate arrays, one for each stack. Each card would be represented by a data item in one of the three stacks. For example, Player1(5) = 3H would mean that the fifth card in player 1's stack is the 3 of hearts. Each stack can contain a maximum of 52 items, as there are 52 cards in the pack. To win, a player's stack of cards or array needs to be full.

A second approach would be to create a single array of 52 items, with the 13 hearts in order from index 1 to 13, followed by the diamonds, spades and clubs. The index for the array is used to represent each card in the pack and the data stored in the array is used to assign each card to a particular stack. For example Card(7) = 0 could mean the 7 of hearts is in the centre stack. Card(24) = 1 could mean the 24th card, which is the 11th card in the diamonds suit, the Jack of diamonds, is in player 1's stack. For player 1 to win requires that all items in the array contain a value of 1.

After considering the processing that will be required, a better structure is discovered that combines aspects of each of the above structures. We use one array containing 52 items, called, say, Cards. The data held in this array is a number that corresponds to each card in the pack. The index for the array indicates the order in which the cards are held in each stack. For example, Cards(18) = 30 means that the 18th card in the stacks is card 30. As 30 – 26 = 4, this card is the 4 of spades. Variables are maintained to store the start position of each of the three stacks within the array Cards. For example, if Stack0 = 1, Stack1 = 12 and Stack3 = 40, then the centre stack is Cards(1) to Cards(11), player 1's stack is Cards(12) to Cards(39), and player 2's stack is Cards(40) to Cards(52). For player 1 to win, Stack1 must have a value of 1 and Stack0 and Stack2 must have values of zero. A small refinement to improve the efficiency is to create an array called Stack that contains 3 items indexed from 0 to 2. Our previous Stack0, Stack1 and Stack2 become Stack(0), Stack(1) and Stack(2).

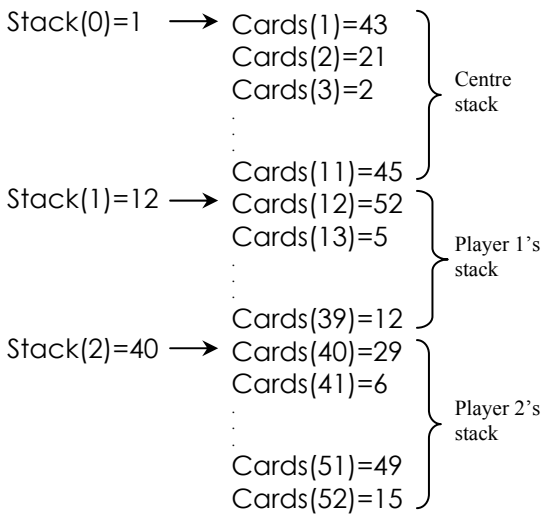


Fig 4.36
Example of the final data structure for 'snap'.

These 2 arrays, Cards and Stack, will be required by most of the modules in the final solution, therefore we declare them as global variables. Individual variables within modules should be local to the particular module.

Algorithms for the card game 'snap'

Using the structure chart in *Fig 4.34*, we can commence designing the algorithms. The main program is quite straightforward as no processing occurs, merely the calling of lower-level modules. Notice that both PlayGame and DisplayResult include the passing of the parameter Result. It is passed from PlayGame and it is passed to DisplayResult as shown in the structure chart. The module DealCards is equally simple: it merely calls the modules ShuffleCards and HalvePack.

```
BEGIN MAINPROGRAM
   DealCards
   PlayGame(Result)
   DisplayResult(Result)
END MAINPROGRAM

BEGIN DealCards
   ShuffleCards
   HalvePack
END DealCards
```

Fig 4.37
Main program and DealCards module for the game "snap".

Let us now work on the ShuffleCards subroutine. In the real game, this involves manually manipulating the cards into a random order. In our simulation we need to assign a unique card number to each element of our array Cards. The numbers randomly assigned must be in the range 1 to 52, and no number can be repeated. If a 5 were to exist in the array twice, this would mean we had two 5 of hearts, which would be unacceptable.

```
BEGIN ShuffleCards
   Cards = randArray(1, 52, 52)
END ShuffleCards
```

Fig 4.38
ShuffleCards algorithm.

Earlier in this chapter we developed three versions of an algorithm to create unique random numbers. We can utilise the best of these algorithms to shuffle our cards. The third attempt at a randArray algorithm was described in *Fig 4.23*. If we call the randArray function with min =1, max = 52 and num = 52 then it will return an array with the numbers 1 to 52 in random order which is precisiely what the shufflecards subroutine needs to do.

To maintain the integrity of the documentation of the original structure chart in *Fig 4.34,* a branch leading from ShuffleCards containing the randArray function needs to be added. Maintaining the integrity of your documentation is vital at all stages of the software development cycle.



GROUP TASK Discussion
With reference to the parameters in both *Fig 4.37* and *Fig 4.38* above, discuss the term 'parameter passing'. Distinguish between byReference and byValue parameter passing as part of your discussion.



GROUP TASK Activity
Design a driver algorithm that could be used to test the ShuffleCards subroutine works as expected. The algorithm should output the values in the array Cards.

The HalvePack routine involves setting the values in the Stack array. This effectively gives player 1 the first 26 cards and player 2 the last 26 cards.



```
BEGIN HalvePack
   Stack(0) = 0
   Stack(1) = 1
   Stack(2) = 27
END HalvePack
```

Fig 4.39
The HalvePack module for the game 'snap'.

Driver
In terms of software development, a driver is a subroutine written to test the operation of another routine. Commonly, drivers set any necessary values, the subroutine to be tested is called, and the results are output.

We now consider the PlayGame subroutine The original structure chart (*Fig 4.34*) provides us with rich information in regard to the logic of this routine. It contains a loop that calls the PlayCard subroutine or the CardMatch subroutine. In other words, either a player plays a card or a match is detected and dealt with. The parameter Result is used to send the overall winner back to the main program. A preliminary design based on this information is shown in *Fig 4.40*. This design is not quite complete but does include all the information from the structure diagram.

A number of questions arise:

- How do we detect matching cards?
- How do we determine when a player has won the game?
- How do we know whose turn it is?

These questions must be solved within the PlayGame subroutine.

```
BEGIN PlayGame(Result)
   WHILE No Result
      IF Match detected THEN
         CardMatch(Winner)
      ELSE
         PlayCard(Player)
      ENDIF
   ENDWHILE
END PlayGame
```

Fig 4.40
Preliminary pseudocode for the PlayGame module based on the structure diagram.

Detecting matching cards requires that we examine the previous card on the centre stack. If the previous card has the same value then we have a match. For example, if player 1 places the 5 of diamonds on the centre stack and then player 2 places the 5 of clubs on the centre stack, we have a match.

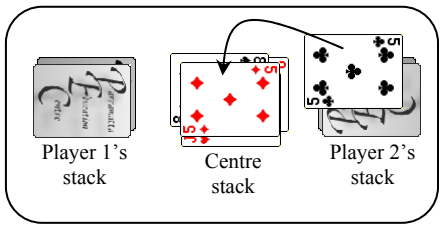


Fig 4.41
A match results when player 2 plays the five of clubs.

The 5 of diamonds has a value of 13 plus 5, which is 18 in our array Cards, and the 5 of clubs has a value of 13 times 3 plus 5, which equals 44. Eighteen divided by 13 leaves a remainder of 5, and 44 divided by 13 also leaves a remainder of 5. Modular arithmetic provides a simpler way of calculating these remainders, as the MOD function is included in most programming environments: 18 MOD 13 = 5, and 44 MOD 13 = 5. In general, CardNumber MOD 13 gives the face value of the card. There is one exception: the Kings, where values 13, 26, 39 and 52 will generate values of 0. This will not affect our game as all we need is for them to be the same.

Remember that our array Cards contains first the centre stack, then player 1's stack, and finally player 2's stack. The previous two cards played, in our example the 5 of clubs and the 5 of diamonds, must be in the first two positions within the Cards array. It is possible that no cards or only one card is/are in the centre stack, so we need to ensure Stack(1) has a value at least 2 more than Stack(0). *Fig 4.42* shows an amended algorithm that incorporates these features.

```
BEGIN PlayGame(Result)
   WHILE No Result
      IF Cards(1) MOD 13 = Cards(2) MOD 13
               AND Stack(1) >= Stack(0) + 2 THEN
         CardMatch(Winner)
      ELSE
         PlayCard(Player)
      ENDIF
   ENDWHILE
END PlayGame
```

Fig 4.42
The revised PlayGame algorithm to detect matching cards.

Determining when a player has won the game involves examining the array Stack. If player 1 has won, then Stack(0) = 0, Stack(1) = 1 and Stack(2) = 0. If player 2 has won then Stack(0) = 0, Stack(1) = 0 and Stack(2) = 1. In fact we need only consider Stack(1) and Stack(2), as the centre stack, Stack(0), must be empty if either of the players has cards in position 1 of the Cards array. We need an efficient decision to do this. If we add the values in Stack(1) and Stack(2) we must get a sum of 1 for there to be a winner. If Stack(1) also contains a value of 1 then player 1 wins, otherwise player 2 wins. Let us develop this as a function that returns 0 meaning no winner, 1 meaning player 1 has won, and 2 meaning player 2 has won. The algorithm FinalWinner in *Fig 4.43* performs this function.

```
BEGIN FinalWinner
   IF Stack(1) + Stack(2) = 1 THEN
      IF Stack(1) = 1 THEN
         fWinner = 1
      ELSE
         fWinner = 2
      ENDIF
   ELSE
      fWinner = 0
   ENDIF
   RETURN fWinner
END FinalWinner
```

Fig 4.43
Function to determine the final winner.

We need to remember to add this FinalWinner function to our structure chart to maintain the accuracy of our documentation. If we were using an integrated CASE tool, the module could automatically be added to the system model when created. An updated version of the PlayGame module is shown in *Fig 4.44*. The FinalWinner function becomes the decision to determine if the game should continue. Once a winner has been determined, the parameter Result is set to the winner's player number.

```
BEGIN PlayGame(Result)
   WHILE FinalWinner = 0
      IF Cards(1)  MOD 13 = Cards(2) MOD 13
               AND Stack(1) >= Stack(0) + 2 THEN
         CardMatch(Winner)
      ELSE
         PlayCard(Player)
      ENDIF
   ENDWHILE
   Result = FinalWinner
END PlayGame
```

Fig 4.44
The revised PlayGame algorithm including calls to the FinalWinner function.

Finally, let us consider whose turn it is to play a card. Player 1 has the first turn, followed by player 2, then player 1 again. Turns continue to alternate until two cards match and a player puts his hand on the centre pile. The player who does this first wins the cards in the centre pile. This winner is returned in the Winner parameter from the CardMatch module. This player has the next turn and then the alternating sequence continues. We need a function that turns a one into a two and a two into a one, so we can easily alternate players. 2 divided by 1 is 2, and 2 divided by 2 is 1, which is precisely what we require. The final PlayGame algorithm is given in *Fig 4.45*.

```
BEGIN PlayGame(Result)
   Player = 1
   WHILE FinalWinner = 0
      IF Cards(1) MOD 13 = Cards(2) MOD 13
               AND Stack(1) >= Stack(0) + 2 THEN
         CardMatch(Player)
      ELSE
         PlayCard(Player)
      ENDIF
      Player = 2 / Player
   ENDWHILE
   Result = FinalWinner
END PlayGame
```

Fig 4.45
The final PlayGame algorithm.



GROUP TASK Discussion
The parameter returned from the CardMatch module has been changed in the final PlayGame algorithm. What effect does this change have on the operation of the PlayGame algorithm?



GROUP TASK Discussion
If a player runs out of cards before a match is encountered, the other player continues to play their cards. Does the PlayGame algorithm deal with this possibility? If not, design a modification to deal with this scenario.

Finally, we will develop an algorithm for the Playcard module. We will leave the remaining algorithms for you to develop.

The PlayCard module receives the player number from the PlayGame module. The top card on the player's stack is removed and added to the top of the centre stack. Changing the values in the Stack array is straightforward; rearranging the cards in the Cards array requires some manipulation.

Let us consider the sample data in *Fig 4.46*. If the PlayCard module is called with the statement PlayCard(2) then it is player 2's turn. As player 2's stack commences at position 32, then card 39 is played. Card 39 is the King of spades. In terms of the array Cards, the data held in Cards(32) is stored in a temporary location, all the items in the array above position 32 move down one place, and finally Cards(1) is assigned the value held in the temporary location, in this example, 39. We then adjust the values of Stack(1) and Stack(2) to reflect our changes. *Fig 4.47* shows our algorithm.

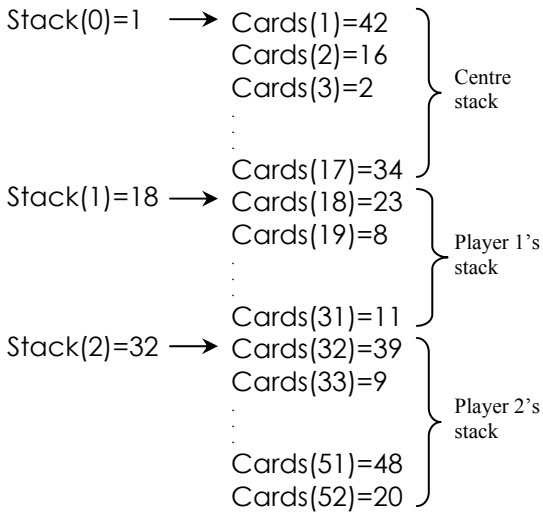


Fig 4.46
Sample data for "snap".

```
BEGIN PlayCard(Player)
   TempCard = Cards(Stack(Player))
   Display TempCard
   MoveItemsDown(Stack(Player))
   Cards(1) = TempCard
   AdjustStacks(Player)
END PlayCard
```

Fig 4.47
The PlayCard algorithm.

We need to create the two subroutines MoveItemsDown and AdjustStacks called by this algorithm. Rather than go into detail, these two subroutines are described in *Fig 4.48*. Remember these modules need to be added to the structure chart to ensure the integrity of the documentation.

```
BEGIN MoveItemsDown(Index)
   Count = Index - 1
   WHILE Count <= 1
      Cards(Count+1) = Cards(Count)
      Subtract 1 from Count
   ENDWHILE
END MoveItemsDown
```

```
BEGIN AdjustStacks(Player)
   Add 1 to Stack(1)
   IF Player = 2 THEN
      Add 1 to Stack(2)
   ENDIF
END AdjustStacks
```

Fig 4.48
Algorithms for the MoveItemsDown and AdjustStacks modules.

Developing Test Data

Developing suitable test data to ensure the correct operation of algorithms is an important aspect of algorithm development. Test data should test every possible route through the algorithm as well as testing each boundary condition. Testing each route makes sure that all statements are correct and that each statement works correctly in combination with every other statement. Testing boundary conditions ensures that each decision is correct. Commonly, a decison will be out by 1 as a result of an incorrect relational operator, for example, Result < 10 rather than Result <= 10. Making sure your test data includes values where Result = 10 will lead to the discovery of the error.



GROUP TASK Discussion
Perform a desk check of the MoveItemsDown subroutine using suitable test data of your own choice. What similarities can you see between this module and aspects of the insertion sort?

Thorough Documentation

Maintaining the integrity of all documentation is vital during the development of algorithms. This is particularly true when a team of developers are working on a software project. Later, when upgrades are under development, the documentation will provide important information to future developers. The revised structure chart for the game 'snap' is shown in *Fig 4.49* to assist in answering the group task that follows.

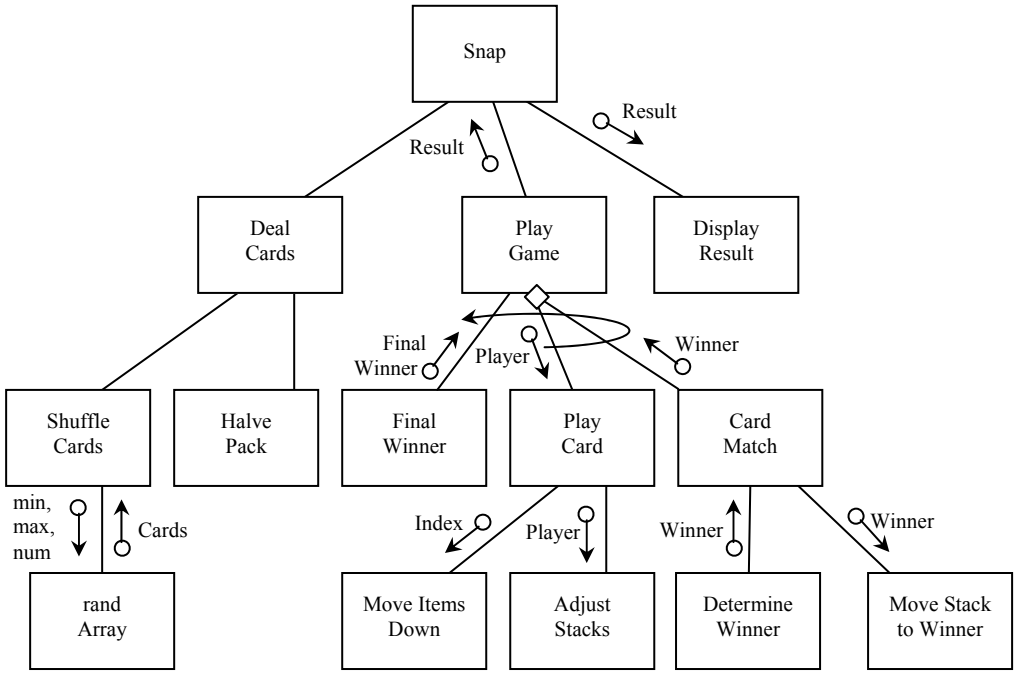


Fig 4.49
Revised Structure chart for the 'snap' simulation.



GROUP TASK Activity
Develop algorithms for the remaining subroutines of the 'snap' card game i.e. CardMatch, DetermineWinner, MoveStackToWinner and DisplayResult. Revise the 'snap' structure diagram to reflect any new subroutines you include in your solution.

SET 4E

1. If RND generates a number in the range 0 up to 1 then INT(RND*16)+2 will generate integers in the range:
   (A) 2 to 16.
   (B) 0 to 15.
   (C) 2 to 17.
   (D) 3 to 19.

2. Test data for algorithms needs to be designed to test:
   (A) all paths through the algorithm and each boundary condition.
   (B) for unexpected inputs.
   (C) for syntax errors in the source code.
   (D) All of the above.

3. A variable used as a parameter in a call to a sub-program:
   (A) must always have the same name.
   (B) must be of the same type as the parameter.
   (C) would normally be a global variable.
   (D) All of the above.

4. The identifiers or names used to declare local variables:
   (A) must be unique within the entire program.
   (B) cause errors if used in more than one subroutine.
   (C) can be used in multiple subroutines without affecting each other.
   (D) cannot be used as parameters in procedure and function calls.

5. Models of the system, such as dataflow diagrams and structure charts:
   (A) should be updated as algorithms are developed.
   (B) remain static once they have been created.
   (C) are used to assist in the training of users.
   (D) need to be revised to describe the logic of each algorithm.

6. 'Algorithm creation is an exact science.' In what sense is this statement correct?
   (A) There is only one solution to each problem.
   (B) There are strict rules that, if followed, will allow the development of a correct solution.
   (C) Correct algorithms are structured in a precise and logical manner.
   (D) All of the above.

7. A data structure is available for access by all subroutines within a system. This data structure can be described as:
   (A) a file.
   (B) local.
   (C) static.
   (D) global.

8. A routine written specifically to test the operation of another routine is called:
   (A) a function.
   (B) a stub.
   (C) a driver.
   (D) a parameter.

Use the following pseudocode to answer questions 9 and 10.

```
BEGIN
   Count = 1
   WHILE Count < 6
      Get Value
      IF Value < Count THEN
         Value = Count
      ELSE
         Count = Count + 1
      ENDIF
      Display Value
   ENDWHILE
END
```

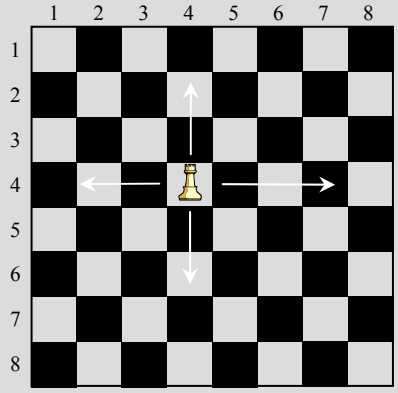
9. Perform a desk check of the above algorithm with the data items 7, 2, 1, 5, 3, 8, 9. What is the output from the desk check?
   (A) 7, 2, 1, 5, 3, 8, 9
   (B) 1, 2, 3, 4, 5, 6
   (C) 7, 2, 3, 5, 4, 8, 9
   (D) 7, 2, 1, 3, 5, 3, 4, 8, 9

10. The best explanation of this algorithm is:
    (A) A pre-test loop goes around 6 times. Each time a value is input, and the smaller of the value and the loop counter is output each time.
    (B) A value is input within a loop that counts from 1 to 6. If the value is smaller than the loop counter then the value of the loop counter is output, otherwise the loop counter is incremented and the value input is output.
    (C) A pre-test loop goes around 6 times. Each time a value is input, the loop counter is incremented and the larger of the value and the loop counter is output.
    (D) A value is input within a loop that counts from 1 to 5. If the value is smaller than the loop counter then the value of the loop counter is output, otherwise the loop counter is incremented and the value input is output.

11. 'The development of a well-structured algorithm is a much more difficult task than implementing the logic in a particular programming language.' Do you agree with this statement? Justify your answer.

Questions 12 to 15 relate to the following scenario:

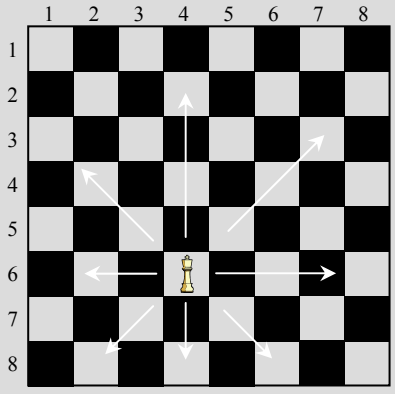
The board game 'chess' uses an 8 by 8 square board. Two players compete against each other using white and black pieces. Without going into the intricacies of the game, each player takes a turn at moving one of his or her pieces. Different types of piece can move in different ways. For example, bishops can move any number of places but only along diagonals. Rooks can move any number of places but only in a straight line. The queen can move both as a bishop and as a rook. Rooks, bishops and queens cannot jump over other pieces when moving. Pieces take the opposing player's pieces by moving onto their squares. You cannot move onto a square containing one of your own pieces.



The rook can move any number of spaces but only in a straight line. This rook is in position Board(4,4).

A software simulation is under construction for this game. The main data structure used is a two dimensional array called Board. This array contains details of the position of all pieces on the playing board. For example, Board(2,4) = BR1 means that row 2 column 4 on the board contains black rook number 1. Board(5,6) = WQ means the white queen is currently in row 5 of column 6.

You, as a developer working for the company developing this product, have been assigned the task of developing a series of routines that ensure legal moves.



The queen can move any number of spaces in any direction. This queen is in position Board(6,4).

12. Create an algorithm for the function called RookCheck that checks that a move performed by a rook is legal. The function has four parameters and returns True or False as its output.

    The format of a function call is as follows:

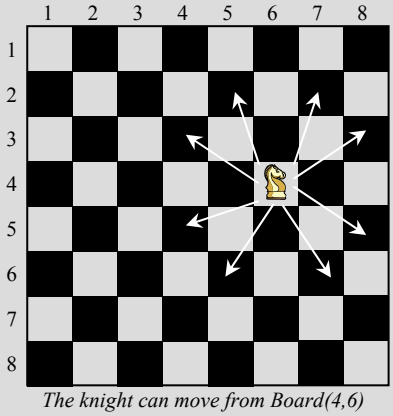
    RookCheck(OldRow,OldCol,NewRow,NewCol)

13. Create an algorithm for the function BishopCheck to check that a move performed using a bishop is legal. The parameters for the function are to be the same as the RookCheck function, developed in question 12.

14. Create an algorithm for the function QueenCheck to check that a move performed using a queen is legal. Use calls to the RookCheck and BishopCheck functions as part of your solution.

15. Knights are able to move in an L-shaped pattern: one space horizontally then two spaces vertically, or two spaces horizontally then one space vertically. The knight is the only piece that may jump other pieces to reach its destination. The diagram at right shows all the possible moves for a knight in row 4 column 6.

    Create an algorithm for the KnightCheck function.



The knight can move from Board(4,6) into any of the squares indicated by the arrows. Knights can jump other pieces.

INTERFACE DESIGN IN SOFTWARE SOLUTIONS

The design of the screens for a software project will be influenced by the nature of the problem. Interactive software products will require more user-friendly screens than those requiring minimal user interaction. For example, most data oriented projects will include data entry screens that will be used for many hours by individual users. The layout and selection of screen elements will have a large effect on the productivity of data entry personnel. Software projects where the final product operates in the background may only require a simple configuration screen. For example: printer drivers, encryption products, virus scanners and many system utilities. These product's screens will only occasionally be used and hence they are of lower importance to the overall product.

If an event-driven approach has been selected for implementation then the screen designs are more likely to be crucial to the operation of the final product. An event-driven program will normally require user input to determine the order of processing. In a sequential program the order of processing has been determined by the developer, hence less selection possibilities are available to the user and screens will be more specific to the processing currently occurring. It is important that the user feels in control of the software rather than feeling controlled by the software.

Let us now examine some major factors to be considered when developing user interfaces:

- Identification of data required.
- Current popular approaches.
- Design of help screens.
- Audience identification.
- Consistency in approach.

IDENTIFICATION OF DATA REQUIRED

Before the task of designing a user interface can commence the data to be included on each screen needs to be determined. The system models, algorithms and data dictionaries will provide this information. Once the required data is ascertained the next task is to decide on the most effective screen element to use to display each data item.

Let us examine a number of common screen elements in terms of the type of data they best represent:

Menus

Menus are used to initiate execution of particular modules within a program. Drop down menus, such as the one in *Fig 4.50*, are used in most GUI based products. Often the items on the menus of a project will correspond to the calls to sub-programs in the mainline of the source code.



Fig 4.50
A typical file menu.

Command buttons

Command buttons are used to select a different path for execution. Often command buttons are used to confirm a task should take place.



Fig 4.51
Command buttons

Tool bars



Fig 4.52
A toolbar showing a selection of icons.

Tool bars are used to quickly access commonly used items. Each tool is represented as an icon. A process is initiated when a particular icon is selected. Tool bars can normally be resized and repositioned. Often toolbars are positioned horizontally at the top or bottom of the screen. Tool bars allow easy access to commonly used functions. The functions available from tool bars are generally also available via the menus.

Text boxes

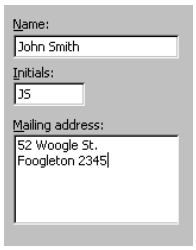


Fig 4.53
A series of text boxes with attached labels.

Text boxes are used to receive input in the form of strings. Text boxes provide no included mechanisms for validating the data entered. The programmer must validate the data entered using code. Numeric data is often input into text boxes as a string and is converted and validated by the program. Text boxes should only be used when other forms of self-validating screen elements are unable to accept the required data. Labels are used to describe the input required to the user.

List boxes

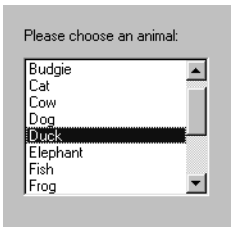


Fig 4.54
A list box forces the user to select from the given options.

List boxes force the user to select from the given options. This screen element is self-validating as it is not possible for the user to enter unexpected input. List boxes can be configured to allow more than one item to be selected. Scroll bars allow the user to scroll through the available options whilst saving space on the screen. The options in a list box can be loaded from an array or input directly by the programmer. The data returned from a list box can be the index of the item selected or the item itself.

Combination Boxes

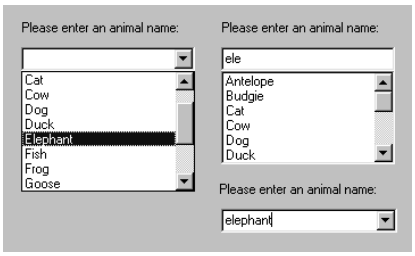


Fig 4.55
Combination boxes combine the functions of a text box with that of a list box.

Combination boxes are often called combo boxes or drop down lists. They combine the functions of a text box with that of a list box. A text box is provided for keyboard entry or the user can select an item from the list box. The list box can be permanently displayed or set to open when the down arrow is selected. Combo boxes are able to force input of one of the items in the list. They can also be set so that new items entered become part of the list.

Check boxes



Fig 4.56
Check boxes are used for the entry of Boolean data.

A check box is used to obtain Boolean input from the user. Clicking a check box causes its value to change from true to false or false to true. This is a self-validating screen element as it is not possible for the user to input inappropriate data.

Option or radio buttons

Radio buttons force the user to select one of the displayed options. It is not possible to select more than one option. In this regard, groups of option or radio buttons are often used to implement multiple selection statements. The index of the option selected is returned to the program. As space needs to be available on the screen for each of the options, this screen element is only used when there are a small number of possible options.

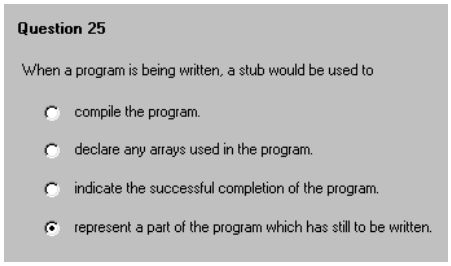


Fig 4.57
Radio buttons force the user to select one of the available options.

Scroll bars

Scroll bars display the position of a numeric data value within a given range. Often they are used to navigate within another screen element. For example, list and combination boxes. Scroll bars give the user a visual interpretation of their position relative to the start and end of the range of possible values. The current position of the scroll bar is returned to the program as a numeric value within a given range.

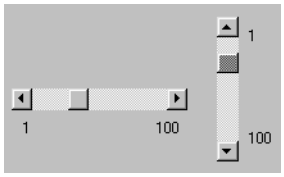


Fig 4.58
Scroll bars are used to represent numeric data within a range.

Grids

A grid is a two-dimensional screen element that can be likened to an array of records. Each row of the grid is used to display individual records and the columns represent each field. The contents of a grid is referenced in a similar fashion to a two-dimensional array, where the row and column numbers are the indexes. Each cell in a grid is really an individual text box. The programmer must validate any data entered into a grid.

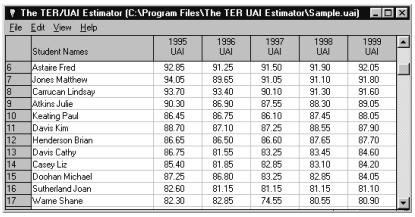


Fig 4.59
A grid is used to present data to the user in the form of a table.

Labels

Labels are used to provide information and guidance to the user. Labels are often used to provide instruction to users in regard to required inputs into other screen elements. The programmer can change the contents of labels. Labels are output screen elements; users cannot input data into labels.

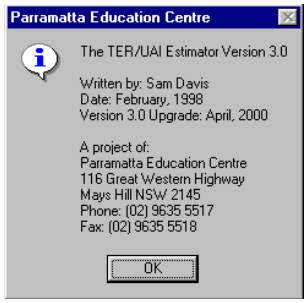


Fig 4.60
A label displaying version details of a software product.

Picture or image boxes

Picture or image boxes are used to display graphics. These are output screen elements. The programmer is able to change the graphic displayed in the box using code.



Fig 4.61
A picture box displaying a company logo.



GROUP TASK Discussion
Examine the interface for a variety of mobile phone apps. Are there any screen elements unique to mobile phone interfaces?

CURRENT POPULAR APPROACHES

Different types of software solution will require different approaches to the design of the user interface. Often the operating system will require a particular design approach. The design of the screen elements is often dictated by the chosen operating system. This is particularly the case with GUI based operating systems.

Internet solutions that will be executed on a large and unknown selection of systems require interface solutions that are suitable for many operating systems.

Software solutions written for dedicated devices, such as mobile phones, microwave ovens, video recorders, etc… will require unique interface designs.

As computers become increasingly more powerful, the detail included in user interfaces has increased. New high-resolution monitors allow greater depth of colour and higher resolution.



Consider the following:

Most applications contain an open dialog as part of the application. Different operating systems have different conventions for the look and feel of open dialogues. This is true in regard to content as well as in regard to the look and feel of the screen's elements.

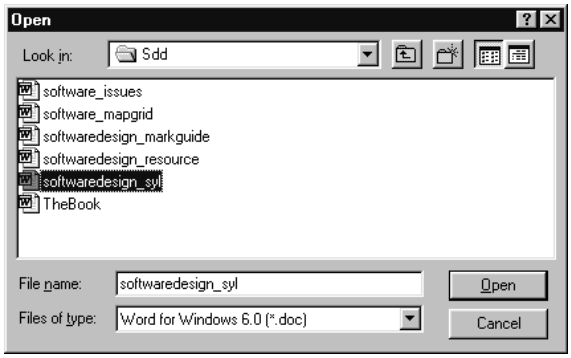


Fig 4.62
An open dialogue from a MS-Windows application.



GROUP TASK Activity
Examine the sample open dialogue above. Identify each of the screen elements included on this dialogue.



GROUP TASK Compare
Find at least two other open dialogues from different operating systems. Compare the designs of these dialogues.



GROUP TASK Discussion
Examine the open dialogues from various applications running under the same operating system. Discuss the reason for the differences in design between these dialogues.

DESIGN OF HELP SCREENS

Help screens are used to assist the user with the operation of the final software product. Ideally, your user interface should be designed in such a way that users can quickly and intuitively learn to use the product. Inevitably however, users will need some assistance in this task.

There are a number of different methods of providing help to users using screens:

- Context sensitive help
- Procedural help.
- Conceptual help
- Tours, tutorials and wizards.

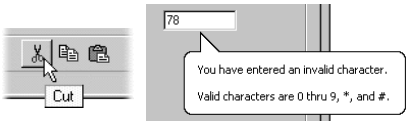


Fig 4.63
Tool tips and balloon text are examples of context sensitive help.

Context sensitive help

By examining the screen element the user is currently working with, it is possible to provide appropriate help. Context sensitive help can take a number of forms.

If the user specifically asks for help, then a new window may be displayed containing information about the user's last action. Often a relevant section from the reference help is an appropriate window to display. It is normally appropriate for the original application screen to remain visible; in other words, the help screen should not take up the whole screen.

Small windows can open to display simple tips relating to the current screen element if the user holds the mouse over an element for more than a few seconds. This type of help should be provided in the form of pop-up windows. Since the user has not specifically asked for help, these windows should disappear once the user moves his or her focus to another element on the screen. They should not require the user to perform any action if the help is not required.

Procedural help

Procedural help should provide concise and accurate instructions on how to complete a specific task. Often a series of steps required to complete the task will be listed. Procedural help aims to explain 'how' a task is accomplished rather than what it is and why it needs to be done.

The help screen used to access procedural help should contain a contents, index and search facility to allow users to quickly locate required information.

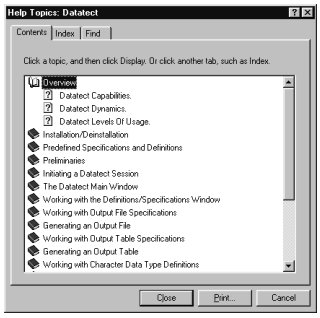


Fig 4.64
Screen allowing access to procedural and conceptual help topics.

Conceptual help

Conceptual help aims to explain information about 'what' and 'why' rather than 'how'. The concepts behind a task should be explained. For example, explaining 'how' to save a file is a procedural task. Explaining 'why' files need to be saved is a conceptual task.

Conceptual help is often accessed from within a procedural help topic. For instance, a description of the steps required to save a file may include a button or hyper-link to explain why it is necessary to save files.

Tours, Tutorials and Wizards

Tours are used to give new users of the product an overview of the product's purpose and basic features. A tour aims to explain the capabilities of a product without detailing the steps needed to accomplish these capabilities. Tours are often multimedia presentations and can be quite separate to the actual software product.

Tutorials aim to provide instruction to users on how to complete specific tasks. Step-by-step instruction is given with examples. These examples may include screen shots from the software product or may lead users through an example task using sample data.

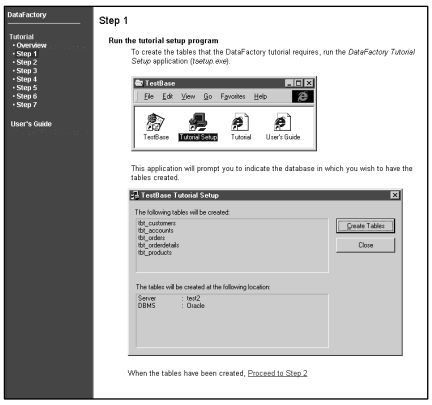


Fig 4.65
A tutorial delivered using a web browser.

Wizards are used to automate complex tasks. The user completes a series of dialogues that gather together the required inputs needed to complete the task. The wizard then completes the task for the user.

Standards and recommendations for the design of user interfaces for tours, tutorials and wizards are provided by many of the leading operating system companies. Many integrated development environments include CASE tools to assist in the creation of screens for these purposes.



GROUP TASK Activity
Examine a number of applications and compile a list of help items from each. Classify each item found either as context sensitive, procedural, or conceptual.

AUDIENCE IDENTIFICATION

The user interface needs to be designed to suit the intended audience. A product written for pre-school children will require quite a different interface to a product designed to perform complex calculations for engineers. An accounting product designed for use by accountants can use the jargon of the finance world. A similar package designed for the general public should not use jargon.

Systems intended for large audiences with unknown levels of expertise may include user interfaces that can be personalised. For example, IBM's OS400 operating system provides help on error messages at three user defined levels: beginner, intermediate and advanced. The wording of error messages changes appropriately to reflect the user's level of expertise.

In chapter 3, we examined design specifications from the users point of view. These specifications provide a framework for the final implementation of the user interface. At this implementation stage, communication with users should continue. Feedback should be sought and acted upon by the development team before the user interface is finalised.



GROUP TASK Discussion
Develop a list of strategies that could be used to identify the audience for a new software product. How can the level of expertise of the audience be best gauged?

CONSISTENCY IN APPROACH

Consistent user interfaces allow users to transfer existing skills and expertise to new products. If the user feels the interface is familiar and predictable, they will be more able to concentrate on the tasks at hand.

User interfaces should be consistent across the application as well as across the platform. As a developer, you should make use of the users expertise with other software products. A consistent interface results in reduced learning curves for users.

Some aspects of interface design where consistency is required include:

- **Names of commands.**

  Users expect to find a cut, copy and paste command. Renaming these as remove, imitate and insert would be unnecessarily confusing for users.

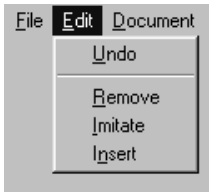


Fig 4.66
Inconsistent command names lead to confusion.

- **Use of icons**

  The save icon is traditionally a floppy disk; to change this to a graphic of a safe would be inappropriate.

- **Placement of screen elements**

  The save command is expected to be on the file menu and the file menu is expected to be the first item on the menu bar. Changing the placement of these elements would reduce the efficiency of the product.

- **Feedback**

  Users expect a response or feedback after they have initiated some action, however the feedback must be consistent across the application. If a find action displays a progress bar as user feedback then other operations that take some time to complete should also display progress bars.

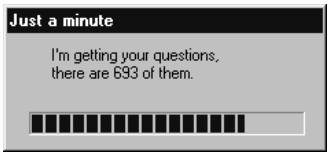


Fig 4.67
Feedback should be provided consistently.

- **Forgiveness**

  Warnings about potential dangerous operations and methods of recovery need to be consistent. Users like to explore an interface to learn by trial and error. The interface should allow for this by permitting users to cancel operations before damage results. Consistency is vital if users are to feel comfortable exploring the interface.



GROUP TASK Investigation
Examine a number of software products in terms of their consistency of user interface design. Compile a class list of any inconsistencies found.

SET 4F

1. The order of processing is determined by the user in the
   (A) sequential approach.
   (B) object-oriented approach
   (C) event-driven approach.
   (D) structured approach.
2. Major factors to be considered when developing user interfaces include:
   (A) identification of data required.
   (B) identification of audience.
   (C) consistency of design.
   (D) all of the above.
3. A screen element used to get Boolean data from the user.
   (A) option or radio button.
   (B) check box.
   (C) list box.
   (D) picture box.
4. A software product opens a print dialog when the print command is initiated. When the save command is selected data is saved without a dialog appearing. This is an example of:
   (A) forgiveness.
   (B) inconsistent feedback.
   (C) inconsistent use of menu items.
   (D) inconsistent use of icons.
5. Procedural help:
   (A) explains the concepts behind a task.
   (B) is always linked to the current work in progress.
   (C) describes why a task is needed and what the task is.
   (D) provides a step-by-step explanation on how to complete a task.
6. 'Making a backup copy of a file provides security should the original become damaged'. This is an example of:
   (A) context sensitive help.
   (B) procedural help.
   (C) conceptual help.
   (D) a tutorial.
7. A commonly used screen element used to confirm a task should take place.
   (A) list box.
   (B) check box.
   (C) command button.
   (D) scroll bar.
8. The level of language used on user interfaces should reflect the:
   (A) operating system standards.
   (B) number of users using the application.
   (C) audience knowledge.
   (D) complexity of the product.
9. Messages that allow users to cancel potentially dangerous operations are addressing which aspect of user interface design?
   (A) Forgiveness.
   (B) Feedback.
   (C) Placement of elements.
   (D) Consistency.
10. A series of screens that leads the user through the basic features available in a product is called a:
    (A) wizard.
    (B) help screen.
    (C) tour.
    (D) tutorial.

11. 'User interfaces should be original and inventive.'
    Do you agree with this statement? Discuss both sides of this argument.

12. Examine the user interface for the printer driver on your home or school computer. List and describe each screen element present on this screen. (A printout of the screen may be helpful).

13. A program requires the entry of personal details; first name, surname, date of birth, sex, address and phone numbers. Create a design for a user interface that could be used to obtain this data.

Mobile phones and now smart phones have become an integral part of our lives. Apart from iPhones, most smart phones currently run a version of the Android operating system. This operating system allows programmers to design and build different user interfaces for different phones and apps.

14. Critically evaluate a number of smart phone user interfaces. Can you suggest any improvements that could be made?
15. Design a user interface for a phone which displays statistics with regard to your contacts. For instance, the interfaces could show contacts in order of total call time, total SMS length, number of calls, number of SMS, etc. Create a storyboard to show the links between screens in your design.

CUSTOMISATION OF EXISTING SOFTWARE SOLUTIONS

Many custom off-the-shelf (COTS) packages allow customisation using a programming language included as part of the COTS package. These languages allow developers to include features not included in the original package.



Consider the following:

A school is developing a reports package to computerise the creation of reports to parents. The school wishes the package to automatically display the text *Above average*, *Average* or *Below average* based on the student's mark compared with the class' average. Marks within 5 marks of the average will be assigned the word *Average*.

The report package is being written in Microsoft Access. This COTS package does not provide the facility to complete this task directly, however it does provide a programming language: Visual Basic for Applications.

An example of the relevant data held in the database tables 'Reports' and 'Classes' is shown in *Fig 4.68*.

Reports

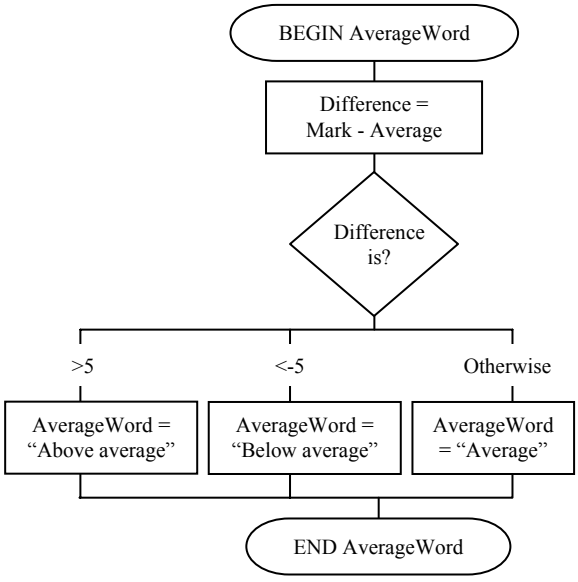
ReportID	StudentID	ClassID	Mark
3158	00001	7DT	83
3162	00001	7EN1	62
3157	00001	7MA2	62
3156	00002	7DT	69
3161	00002	7EN1	83
3163	00002	7MA2	72
3159	00003	7EN2	44
3160	00003	7DT	
3164	00003	7MA2	70

Classes

ClassID	CourseID	Average
7MA1	MA	65
7DT	DT	67
7EN1	EN	66
7MA2	MA	60
7EN2	EN	63

Fig 4.68
Sample data from a school's reports package.

An algorithm is developed to perform this task. This algorithm is then implemented as a function using Visual Basic for Applications. The function AverageWord then becomes part of the Reports package and can be used to generate the required text on the final reports.




```
Public Function
AverageWord(Mark,Average) As String
   Difference=Mark-Average
   Select Case Difference
      Case Is > 5
         AverageWord="Above average"
      Case Is < -5
         AverageWord="Below average"
      Case Else
         AverageWord="Average"
   End Select
End Function
```

Fig 4.69
Algorithm and Visual Basic for Applications code for the AverageWord function.

SELECTION OF LANGUAGE TO BE USED

Algorithms should be written in such a way that they could be implemented in a variety of languages. The most appropriate programming language should be selected after considering each of the following criteria:

- Is the programming logic driven by the user or by the programmer?
- Does the language provide for all the required features of the solution?
- What hardware and operating system ramifications are there?
- Is a graphical user interface (GUI) required?
- What is the experience of the developers?

All these questions need to be considered carefully.

IS THE PROGRAMMING LOGIC DRIVEN BY THE USER OR BY THE PROGRAMMER?

The answer to this question will determine whether the language selected should use an event-driven approach or a sequential approach. In the Preliminary course, we studied the major differences between these two approaches.

Event-driven languages allow the user to choose the order in which processing occurs. For example, a Word Processor is clearly an event-driven piece of software. The user decides in which order tasks are accomplished. They can enter all the text and then choose to edit and format the text, or they can edit as they go and format at the end. A Word Processor allows the user to alter the order of processing, as each process is initiated by the user. In modern GUI environments, events are usually initiated by selecting an item from a drop-down menu. Once the menu item is selected, the particular module of code associated with that event is executed.

Sequential languages ensure that the software executes in a distinct order. The programmer determines the order of processing when writing the software. The user must follow a distinct route through the software. For example, the game of ‘snap’ examined in the previous section, must be played according to a set of rules that determine a strict sequence of processing. Players cannot play out of turn; the game follows a strict sequence of events.

Many problems involve a combination of approaches. For the game of ‘snap’ we may wish either player to be able to terminate the game at any time. If this is the case, we would provide a menu item to end the game that is always available. A player initiating the end game option is effectively initiating an event. The program responds to this event by running a module that halts the game. Once an event, say to start the game of ‘snap’, is initiated, then a distinct sequence of events must be completed before another event can be begun. In effect, event-driven software combines a number of smaller, sequential modules.

DOES THE LANGUAGE PROVIDE FOR ALL THE REQUIRED FEATURES OF THE SOLUTION?

The reason so many programming languages exist is that solutions to different problems require different features within languages. This has given rise to the development of a vast assortment of languages. Solutions requiring many low-level operations will best be implemented in a quite different language to that used for the development of a new web page. Financial software would be implemented in a different language than for a product based on the processing of text.

Careful consideration of features required to implement a solution, and an intelligent selection of an appropriate language, will result in reduced development times and more efficient final products.

WHAT HARDWARE AND OPERATING SYSTEM RAMIFICATIONS ARE THERE?

Many programming languages are designed for particular operating systems. As operating systems are specific to particular types of hardware, the programming language selected will most likely have hardware ramifications. This problem is particularly prevalent when using COTS (Customised off-the-shelf) packages as development tools. Most COTS packages are designed to operate under a specific operating system. A product developed using a Unix-designed programming environment will normally require significant changes to operate in a Macintosh environment.

If a language is used that does not support the system settings provided in modern operating systems, then drivers for different peripheral devices may need to be written. For custom products written for a particular known client, this situation may be acceptable, but for products developed for a large and unknown customer base this situation is anything but acceptable. For example, it would not be economically feasible to write drivers for every available printer on the market. It is far wiser to use a language environment that can utilise the operating system's printer drivers and settings.

IS A GRAPHICAL USER INTERFACE (GUI) REQUIRED?

For personal computer-based applications the use of a standard GUI is almost obligatory. The majority of users will be accustomed to the familiar features of their installed GUI-style operating system. Custom solutions developed for particular clients are another matter. A software solution developed for a new model of DVD player will use a proprietary user interface designed specifically for the particular player. Software products that operate with minimal user interaction are often best developed without the extra overhead of a GUI interface. For example, a software encryption product for the military runs in the background, with only very limited user interaction required. The users, who have a high-level of expertise, will only interact with the product to alter configuration settings. The use of a GUI would reduce the efficiency of this product.

WHAT IS THE EXPERIENCE OF THE DEVELOPERS?

The first programming language learnt by a developer is always the most difficult. As developers become proficient in further languages, the learning curve flattens. Nevertheless, learning a new programming language is a difficult and time-consuming task. Before deciding on a language for implementation, the experience and expertise available within the software development team need to be examined. If a new language is used, then development times will need to be extended.

FACTORS TO BE CONSIDERED WHEN SELECTING THE TECHNOLOGY TO BE USED

Throughout this chapter our focus has been on the correctness, efficency and performance of algorithms. Clearly enormous performance improvements can be achieved when algorithms are improved, such as changing an algorithm from $O(n^2)$ complexity down to $O(n)$ complexity. However algorithms do not govern performance alone. The various other technologies including hardware and software will of course have a significant impact. Many software projects are part of complete new information systems, therefore the system designers have control over all the software and hardware components that will be used. For software only projects the selection of an appropriate programming language is within the control of most software developers as is the target operating systems and some general or recommended hardware requirements.

Performance requirements

Comparing the performance of different hardware components, such as different CPUs or different HDDs is a difficult task. There are many factors that should be considered and not all factors will be relevant for all systems. Performance is not just about speed, it also includes the fault tolerance of the device. A fast device that fails within weeks is a poor performer. Can it recover from power spikes and power failures? What occurs when the device is under load, such as many users trying to read from a HDD at the same time. The specific performance requirements should be established so that tests can be developed to ensure these requirements are met.

Benchmarking



Benchmarking
The process of evaluating something by comparison to a standard. Benchmarking enables competing products to be fairly compared.

Benchmarking a product is the process of evaluating the product by comparing it to an established standard. When other competing products are also compared to the standard then it is possible to compare the competing products. For computers, software and computer components standards are commonly developed which assess performance requirements found to be important to many users. For example, the ability to render 3D graphics in high definition at a particular speed would be a relevant performance requirement for gamers. A standard to assess this requirement might specify a particular game or utility which should be executed whilst the computer is also running a web browser and virus scan software. Often the benchmarking software is able to record techincal details of the variables in question so that a fair mathematical and statistical evaluation can be made.



GROUP TASK Research
PassMark is an example of benchmarking software, but there are many others. Research and describe the nature of the tests performed by benchmarking software applications.



GROUP TASK Practical Activity
Using an algorithm you have coded, such as a sort or maybe the randArray algorithm, create a test application which records the execution time of the code. Run the test on different systems under different conditions and evaluate the results.

CHAPTER 4 REVIEW

1. Which of the following lists is in the correct order of creation?
   (A) System models, algorithms, data structures.
   (B) Data structures, system models, algorithms.
   (C) System models, data structures, algorithms.
   (D) Algorithms, system models, data structures.

2. Which of the following statements is true?
   (A) Linear searches are generally faster than binary searches.
   (B) The data items must be sorted before a linear search.
   (C) Binary searches discard half the list after each comparison.
   (D) A binary search will examine every data item in the list.

3. At least one item must be in its correct final position after a single pass through the data items when using:
   (A) an insertion or bubble sort.
   (B) a selection or insertion sort.
   (C) a selection or bubble sort.
   (D) a selection sort only.

4. Adding a new string into the middle of an existing string could be called:
   (A) extracting.
   (B) deleting.
   (C) inserting.
   (D) copying.

5. Which sort does not involve swapping data items?
   (A) Insertion sort.
   (B) Selection sort.
   (C) Bubble sort.
   (D) Both insertion and selection sorts.

6. Finding the largest value in an unsorted list of items is:
   (A) similar to one pass of a selection sort.
   (B) the main concept behind an insertion sort.
   (C) accomplished using a binary search.
   (D) possible only when the data is numeric.

7. Adding one string to one or more other strings is called:
   (A) concatenation.
   (B) inserting.
   (C) combining.
   (D) ampersand.

8. A data structure where each individual data item may have a different data type is:
   (A) an array of records.
   (B) a record.
   (C) an array.
   (D) a multi-dimensional array.

9. Records that need to be accessed directly would be stored in a:
   (A) sequential file.
   (B) relative file.
   (C) text file.
   (D) binary file.

10. Languages that use an event-driven approach contain all the following features EXCEPT:
    (A) Programs written in them have a distinct start and finish.
    (B) The user controls the order of processing.
    (C) They often use a GUI.
    (D) Subroutines are executed as a result of some user action.

11. Often a series of records need to be sorted by two fields. For example, it is common to sort names by surname, and then within each surname by first name. Can this be accomplished by applying one of the sorts studied, twice? Explain your answer in terms of the insertion, selection and bubble sorts.

12. Read the following scenario and create an algorithm to complete the task:

    Throwing two dice always results in a total between two and twelve, however each total is not equally likely. For instance, there is only one way of throwing a total of two: you must throw a double one. However, a total of four can be obtained in three ways: a double two, a one and a three or a three and a one. We could work out the theoretical probability for each total, however what if we threw the two dice a million times? How many times would each total come up?

    Create an algorithm to simulate throwing two dice a million times. Your algorithm is to output the number of times each total is obtained.

13. Internet search engines allow us to enter a list of words separated by spaces. The search engine then looks for web sites that include any combination of the words entered. The search engine examines the total string you have entered and splits it into individual words based on the spaces separating the words.

    Create an algorithm that receives a string of words separated by spaces. The algorithm is to split this string into its component words and store the words in an array.



Examine the algorithms at right to answer questions 14 and 15.

14. Describe the nature and data type of each of the following: IsMatch, ClientName, ClientID, ProductID, FindClientID, Index, Client, Clients, OrderExists, Order.

15. What do you think is the purpose of these algorithms? Explain.

```
BEGIN IsMatch(ClientName,ProductID)
   ClientID=FindClientID(ClientName)
   IsMatch=OrderExists(ClientID,ProductID)
END IsMatch

BEGIN FindClientID(Client)
   Set Index to 1
   WHILE More records
      IF Clients(Index).Name=Client THEN
         FindClientID=Index
      ENDIF
      Add 1 to Index
   ENDWHILE
END FindClientID

BEGIN OrderExists(ClientID,ProductID)
   IF Order(ClientID,ProductID) is blank THEN
      OrderExists=False
   ELSE
      OrderExists=True
   ENDIF
END OrderExists
```

In this chapter you will learn to:

- identify an appropriate language to solve a particular problem
- recognise the appropriateness of either a sequential or event-driven approach to solve a particular problem
- develop syntactically correct code to solve a problem in a given language
- interpret metalanguage definitions for commands in a selected language
- produce syntactically correct statements using the metalanguage definitions
- produce a generic metalanguage definition for a set of syntactically correct statements that use the same command
- implement a solution from a complex algorithm using syntactically correct statements
- explain the use of tokens and the role of the parsing process during the translation of source code to machine code
- recognise that machine code is the only code able to be executed by a computer
- identify the most appropriate translation method for a given situation
- use the features of both a compiler and an interpreter in the implementation of a software solution
- recognise, interpret and write machine code instructions for a problem fragment
- employ good programming practice when developing code
- justify the use of a clear modular structure with separate routines to ease the design and debugging process
- differentiate between types of errors, recognise the cause of a specific error and determine how to correct it
- effectively use a variety of appropriate error correction techniques to locate the cause of a logic error and then correct it
- produce user documentation (incorporating screen dumps) that includes a user manual, a tutorial, online help
- differentiate between types of user documentation
- identify the personnel who would be likely to use the different types of documentation
- produce technical documentation for an implemented software solution
- recognise the need for additional hardware
- identify potential compatibility issues for a newly developed software solution
- recognise the implications of emerging technologies for the developer in terms of the code written to make use of these technologies
- recognise the implications of emerging technologies for the code development process

Which will make you more able to:

- explain the interrelationship between hardware and software
- differentiate between various methods used to construct software solutions
- describe how the major components of a computer system store and manipulate data
- explain the implications of the development of different languages
- explain the relationship between emerging technologies and software development
- identify and evaluate legal, social and ethical issues in a number of contexts
- construct software solutions that address legal, social and ethical issues
- apply appropriate development methods to solve software problems
- apply a modular approach to implement well structured software solutions and evaluates their effectiveness
- apply project management techniques to maximise the productivity of the software development
- create and justify the need for the various types of documentation required for a software solution
- selects and applies appropriate software to facilitate the design and development of software solutions
- communicate the processes involved in a software solution to an inexperienced user
- use and describe a collaborative approach during the software development cycle
- develop and evaluate effective user interfaces, in consultation with appropriate people

In this chapter you will learn about

Implementation of the design using an appropriate language

- the different programming languages and the appropriateness of their use in solving different types of problems
- construction of syntactically correct code that implements the logic described in the algorithm

Language syntax required for software solutions

- use of EBNF and railroad diagrams to describe the syntax of statements in the selected language

The need for translational to machine code from source code

- translation methods in software solutions including compilation and interpretation
- advantages and disadvantages of each method
- steps in the translation process
  - lexical analysis including token generation
  - syntactical analysis including parsing
  - code generation

The role of machine code in the execution of a program

- machine code and CPU operation
  - instruction format
  - use of registers and accumulators
  - the fetch–execute cycle
  - use of a program counter and instruction register
- execution of called routines
- linking, including use of DLLs

Techniques used in developing well-written code

- the use of good programming practice, including:
  - a clear and uncluttered mainline
  - one logical task per subroutine
  - use of stubs
  - appropriate use of control structures and data structures
  - writing for subsequent maintenance
  - version control and regular backup
  - recognition of relevant social and ethical issues
- the process of detecting and correcting errors, including:
  - types of error
    - syntax errors and logic errors
    - runtime errors, including arithmetic overflow, division by zero, accessing inappropriate memory locations
  - methods of error detection and correction
    - use of flags
    - methodical approach to the isolation of logic errors
    - use of debugging output statements
    - peer checking, desk checking, structured walkthrough
    - comparison of actual with expected output
- the use of software debugging tools, including use of breakpoints, resetting variable contents, program traces, single line stepping

Documentation of a software solution

- forms of documentation, including:
  - log book
  - user documentation, including user manual, reference manual, installation guide, tutorial, online help
  - technical documentation, including systems documentation, algorithms, source code
- use of application software including CASE tools to assist in the documentation process
- recognition of relevant social and ethical issues

Hardware environment to enable implementation of the software solution

- hardware requirements including minimum configuration, possible additional hardware, appropriate device drivers or extensions

Emerging technologies

- the effect of emerging hardware and software technologies on the development process, such as iPhone, Wii remote, handheld communication devices, scanning pen, biometric devices, multi-point surface software, radio frequency identification (RFID), social networking software.

5

IMPLEMENTATION OF SOFTWARE SOLUTIONS

Implementation of software solutions is the third stage of the software development cycle. During this stage of development the source code is written and tested. That is, the plans and designs formulated in stage two are implemented in a programming language.

We first briefly examine a variety of different programming languages with an emphasis on their appropriateness for solving different types of problems. The syntax of high-level programming languages is then examined using metalanguages such as EBNF and railroad diagrams. This information will allow for the creation of syntactically correct source code.

The source code is translated into machine code. We examine this process of translation including compilation and interpretation. The machine code or object code is then executed by the CPU. This process of execution is studied, with particular reference to the components and actions occurring within the CPU. These actions result in the execution of machine code commands.

Program development techniques are examined. This section describes how well structured, modularised and documented source code should be created. It also explains methods of error detection and correction.

Documentation of the final solution for various audiences is vital for the ongoing support of a software product. We examine a number of different forms of documentation for users and future developers.

Software solutions must operate with hardware and existing software. We consider the hardware and software requirements of new software products. Finally, we examine some emerging technologies and trends likely to affect future software development.

CHOOSING AN APPROPRIATE PROGRAMMING LANGUAGE

There are numerous factors to consider when selecting an appropriate programming language for a specific project. For some projects there will be little choice due to the nature of the targeted environment and device. When Apple's iPhone and iPad were released programmers were limited to Objective C, C++ or C programming languages and similarly Android developers where limited to Java. Although (in 2012) these remain the languages of choice for the development of most smart phone apps many other languages can be used. For software applications running on desktops the range of languages is extensive, however rightly or wrongly it is often the experience of the developers that becomes the overriding selection criteria. Web-based software commonly contains two separate elements – the server side code and the client side code which often runs within the user's browser. In most cases one language is used for the server side and quite a different language is used for the client side code. For instance, PHP is commonly used on the server and JavaScript on the client side.

When choosing a programming language a variety of different criteria should be considered including:

- The nature of the project. Different languages are suited to the solution of different types of problems. An artificial intelligence application will be suited to a different language compared to software to run an online store. The best language to write a low level hardware driver will be very different to the best language to write a platform game. Research is required to determine the most suitable language for the type of software project.
- The intended environment including hardware, operating system and network use has a large impact on the most suitable language. There are interpreted languages designed for use on web servers and others which operate within client web browsers. Some languages compile down to an intermediate code (such as Java's byte code) and will execute on multiple different operating systems. Other languages or language variants are designed for smart phones or tablets.
- The experience of the programmers. It takes a significant amount of time and effort to learn a new language and to become an expert requires experience which can only be obtained over time. Therefore, it is often better to select a language the developer's are already familiar with rather than choose a new language despite the new language being more appropriate for a particular project.
- Event driven or sequential approach. Software applications which require a graphical user interface (GUI) often use an event driven approach. This means subroutines within the software execute in response to events initiated by the user, such as clicking a button or selecting from a menu. Other events such as an incoming network message or a sensor detecting movement are also events the system can detect. The design of event driven code is different to sequential code as there is no definite start and end. A sequential program has a single start, the processing commences and then finally the program ends. The code controls the order of execution, which includes directing the user through the logic. Event driven programs are controlled and led by the user. As the user initiates commands the program runs the code associated with that event.
- Maintainability. Does the language encourage or enforce good practice in terms of well structured and self contained modules which are easy to maintain and reuse? Is there a large community of programmers and a large library of code which can be examined for ongoing support?



Consider the following:

Over a number of years the following languages have been consistently ranked within the top 20 according to a variety of different surveys:

- Java
- JavaScript
- C
- C#
- C++
- Objective-C
- PHP
- Visual Basic (including VB.NET)
- Python
- Delphi
- Pascal
- Lisp
- Ruby
- Perl



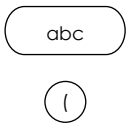
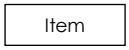
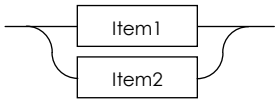
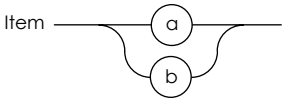
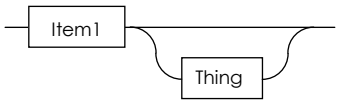
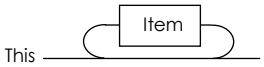
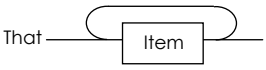
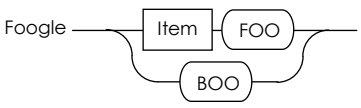
GROUP TASK Research
Each class member is to research the general nature of a few of the above languages. Share a summary for each language with the class.

LANGUAGE SYNTAX REQUIRED FOR SOFTWARE SOLUTIONS

In this section, we examine the syntax of programming languages. The syntax of programming languages is described using metalanguages. In the Preliminary course, we examined EBNF and railroad diagrams. We will use these metalanguages to describe the syntax of commands used to define and use multi-dimensional arrays, arrays of records, files and random number generators.

USE OF EBNF AND RAILROAD DIAGRAMS

Before we begin using these metalanguages to describe the syntax of programming languages, let us revise the syntax of the metalanguages themselves:

Interpretation	EBNF example	Railroad Diagram example
Terminal Symbol for a reserved word. BEGIN is a reserved word.	BEGIN	
Terminal symbol for a literal. The characters abc are written as they appear. Quotes are used to enclose symbols used by the metalanguage.	abc "("	
Non-terminal symbol. Item is defined elsewhere.	<Item>	
"or" a choice between two alternatives. Either Item1 or Item2.	<Item1> | <Item2>	
"is defined as".Item can take the value a or b	Item=a | b	
Optional part. Item followed optionally by a Thing.	<Item>[<Thing>]	
Possible repetition. This is an Item repeated zero or more times.	This={<Item>}	
Repetition. That is an Item repeated one or more times.	That=<Item>{<Item>}	
Grouping. A Foogle is an Item followed by the reserved word FOO or it is the reserved word BOO.	Foogle=(<Item>FOO) | BOO	



GROUP TASK Activity
Create an EBNF and railroad diagram for a Boogle. A Boogle always begins with an Item, followed by one of more Foogles and can either end with a That or a This. Create a series of five legitimate Boogles using your metalanguage descriptions.

METALANGUAGE DESCRIPTIONS OF PROGRAMMING LANGUAGES

Let us examine metalanguage descriptions for defining and using multi-dimensional arrays, arrays of records, files and random number generators in a variety of high-level languages.

Declaring multi-dimensional arrays and arrays of records

In general, declaring a variable for use within a program involves two stages. Firstly, if the data type or structure does not already exist in the language then it must be defined. Secondly, a variable of the required type is declared.

EBNF diagrams to perform these actions are described below:

Visual Basic .NET

The Structure statement is used to create a user defined data type.

```
AccessModifier = Dim | Public | Private

Identifier = <Letter> { <Letter> | <Digit> | _ }

DimStatement = <AccessModifier> <Identifier> [ "("<PositiveInteger>")" ] As <Type>

StructureStatement = <AccessModifier> Structure <Identifier>
                                <DimStatement>
                                { <DimStatement> }
                             End Structure
```

The Type non-terminal element above can be any of the available types included within Visual Basic (String, Integer, Double, Boolean, etc.) or it can be a user defined type that has been declared using a Structure statement.

The description of the DimStatement above indicates that arrays can be declared directly without using the StructureStatement. To declare an array the identifier is followed by the upper bound in brackets, such as Private MyArray(4) As String to declare a string array called MyArray with 5 elements (subscript 0 to 4). However, record data structures require definition using a StructureStatement and then declaration using a DimStatement.

Pascal

The TYPE statement is used to create a user defined data type.

```
DefineDataType=TYPE <varname>=(ARRAY"["<range>{,<range>}"]"OF<type>) |
                    (FILE OF <type>) |
                    (RECORD<varname>:<type>{;<varname>:<type>}END);
```

The VAR statement is used to declare variables of the required types.

```
DeclareDataType=VAR <varname>{,<varname>}:<type>;
                     {<varname>{,<varname>}:<type>;}

varname=<letter>{<letter> | <digit>}
range=<constant>..<constant>
```



GROUP TASK Discussion
Examine the above EBNF diagrams. Discuss the essential differences between the two languages, particularly in regard to declaring arrays.

Note: The EBNF and railroad diagrams used in this chapter explain features necessary for this course. The actual EBNF and railroad productions for these statements include many more features which have been omitted for purposes of clarity.

Using multi-dimensional arrays



Consider the following:

A program requires a multi-dimensional array called Products to be available to all procedures and functions.

The Products array is to have three indexes. The first index ranging from 0 to 10 is used to represent the Supplier. There is a maximum of 11 suppliers. The second index represents one of the five warehouses where the product is stored. The third index determines individual products. It is unlikely there will be more than 100 products from a particular supplier stored in a particular warehouse. Each data item stored in the array will be a string with a maximum of 20 characters.

For example, Products(2, 3, 41)="Widget" means that Widgets are from supplier number 2 and are stored in warehouse 3. Widgets are the 40th product from supplier 2 to be stored in warehouse 3. Note that each index begins at zero hence there is a supplier number 0, a warehouse number 0 and a product 0.

Let us create the products data structure in both Visual Basic and Pascal using the EBNF diagrams from the previous page.

Visual basic

A possible definition for our Products variable could be:

```
Public Products(10, 4, 99) AS String
```

This statement is best placed in the declaration section of a module. The Public keyword ensures that Products is a global variable available to all procedures and functions. The first index has a range of 0 to 10, in Visual Basic 0 is the default for the lower bound so need not be included.

Pascal

A possible definition for our Products variable could be:

```
TYPE
   ProductType=ARRAY[0..19] OF Char;
   ProductArrayType=ARRAY[0..10,0..4,0..99] OF ProductType;
VAR
   Products      : ProductArrayType;
```

In Pascal, strings are not standard data types and must be defined as arrays of characters. Once the array of characters has been declared, it can be used to define other data structures.



GROUP TASK Investigation
Examine another programming language and develop declaration statements for the above Products array.

One of the modules that uses the Products array is described in the IPO Chart below:

IPO CHART

Client: Choog Soft Pty. Ltd.
Project: Warehousing System
Module: NextProductNumber

Date: 29/2/2001
Programmer: Melissa Davis
Page: 15 of 28

Input	Process	Output
Supplier Number, Warehouse Number	Examine each product in turn with Supplier Number and Warehouse number.	
	If Product is empty then this Index provides the next available Product Number.	
	Return the product number to the calling module. Return –1 if no space available.	NextProductNumber

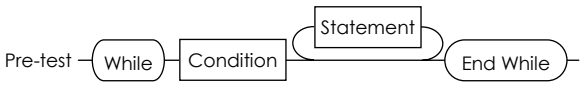
Fig 5.1
IPO Chart for the FindProductNumber function.

Melissa has already developed an algorithm for this function. Her algorithm is shown in *Fig 5.2*.

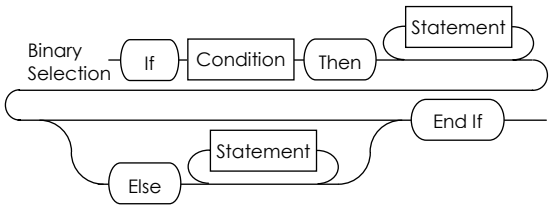
Let's implement the NextProductNumber function in both Visual Basic and Pascal.

Visual Basic .NET

A railroad diagram for the pre-test loop structure in Visual basic is:



For binary selection the Visual Basic railroad diagram is:



In Visual Basic the function could be written as follows:

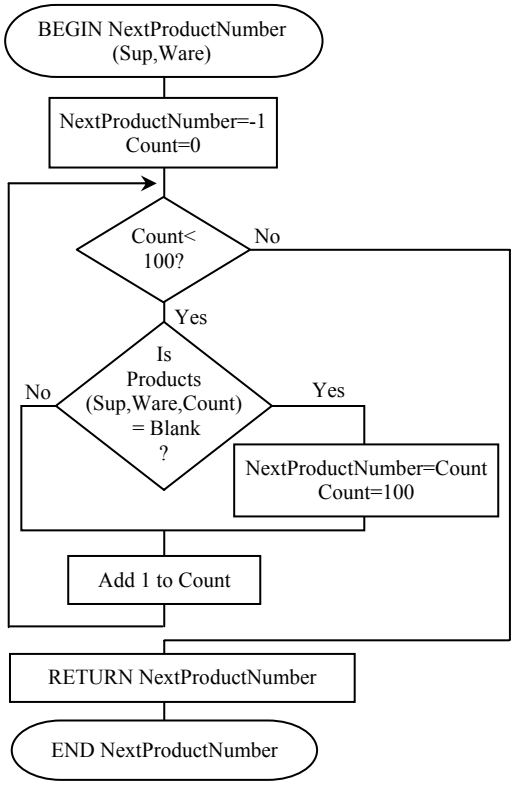


Fig 5.2
Algorithm for the FindProductNumber function.

```
Function NextProductNumber(ByVal sup As Integer, ByVal ware As Integer) As Integer
   Dim count As Integer
   NextProductNumber = -1
   count = 0
   While count < 100
      If Products(sup, ware, count) = "" Then
         NextProductNumber = count
         count = 100
      End If
      count = count + 1
   End While
   Return NextProductNumber
End Function
```

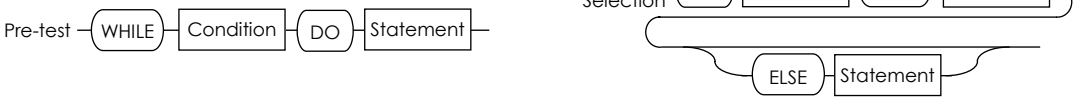


GROUP TASK
Activity

Create some sample data for the products array. Complete a desk check of the Visual Basic code. Explain, in words, the processing taking place. This function is similar to one of the searches from chapter 4; explain the similarities.

Pascal

Railroad diagrams for the pre-test loop and binary selection structures in Pascal are:



For those readers not familiar with Pascal, a statement can be made up of multiple statements. When multiple statements are used then the keywords BEGIN and END are used to group the statements. Semi-colons are used to separate each statement within a multiple statement.

In Pascal the function could be written as follows:

```
FUNCTION NextProductNumber (Sup : Integer;
                            Ware : Integer) : Integer;
VAR
   Count : Integer;
BEGIN
  FindProductNumber:=-1;
  Count:=1;
  WHILE Count<=100 DO
      BEGIN
        IF Products[Sup,Ware,Count]="" THEN
            BEGIN
              FindProductNumber:=Count;
              Count:=100
            END;
        Count=Count+1
      END
END;
```



GROUP TASK Discussion

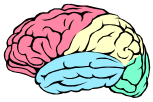
The compiler ignores the spacing and indenting in the above Pascal code. The code would operate exactly the same if no spacing or indenting was used. Discuss the reasons for including spacing and indenting in source code.



GROUP TASK Investigation

Examine another programming language and develop the NextProductNumber function in this language.

Using arrays of records and files



Consider the following:

Imagine that you work for Bizzy Soft Pty. Ltd. a software development company that writes business applications. You are currently working on a product to manage customer details for businesses.

The current project requires an array of records called Customers. This array of records needs to store up to 500 customers. The details for each customer include their Surname, First Name, Sex and Email address. Surnames and First Names have a maximum length of 20 characters, Email addresses up to 30 characters and the Sex can hold one of two values.

You have been allocated the task of writing two subroutines for this product. One to save all the customer records to disk and another to retrieve them from disk back into the Customers data structure.

Let us create the Customers data structure in both Visual Basic .NET and Pascal.

Visual Basic .NET

```
Public Structure CustomerType
   Public Surname As String
   Public FirstName As String
   Public Sex As Boolean
   Public Email As String
End Structure

Public Customers(500) As CustomerType
```

However the above CustomerType definition creates strings of variable length. This will work correctly if we use a sequential file, however for a relative (or random) access file we need to create fixed length strings so that each record will always be precisely the same length. The following definition creates fixed length records:

```
Public Structure CustomerType
   <VBFixedString(20)> Public Surname As String
   <VBFixedString(20)> Public FirstName As String
   Public Sex As Boolean
   <VBFixedString(30)> Public Email As String
End Structure

Public Customers(500) As CustomerType
```

Pascal

```
TYPE
   CustomerRecType=RECORD
     Surname    : ARRAY[1..20] OF Char;
     FirstName  : ARRAY[1..20] OF Char;
     Sex        : Boolean;
     Email      : ARRAY[1..30] OF Char
   END;

   CustomerType=ARRAY[1..500] OF CustomerRecType;

VAR
   Customers    : CustomerType;
```



GROUP TASK Investigation
Examine another programming language and develop declaration statements for the above Customers data structure.

We now need to develop the sub-programs to store and retrieve the Customer's records from a disk file. As the developer, you have already created an algorithm to complete the storage task (see *Fig 5.3*). Retrieving the records will use the identical algorithm except instead of writing the records to the file we will be reading them from the file.

```
BEGIN StoreCustomers (FileName)
   Open Filename for output
   Count=0
   WHILE Count<=500
      Write Filename from Customers(Count)
      Add 1 to Count
   ENDWHILE
   Close Filename
END StoreCustomers
```

Fig 5.3
Algorithm to store customer details to file.

Let us now examine the syntax of statements required to complete these file access tasks in Visual Basic.

Visual Basic .NET

The required EBNF diagrams are:

The FileOpen statement is used to open (or create) a file and prepare it for input or output:

OpenFile = FileOpen "(" <FileNumber>, <FileName>, <Mode>, [<Access>], [<Share>], [<RecordLength>] ")"

Mode = Append | Input | Output | Random

Access = Read | Write | Read/Write

Share = LockRead | LockWrite | LockRead/Write | Shared

RecordLength = <integer>

FileName is used to specify the full path including the filename to the file. Mode can be Input, Append or Output for sequential files or Random for random access files. Filenumber is an integer between 1 and 511. Each file opened must have a unique file number. The next free file number can be obtained using the FreeFile function. RecordLength is required for random access files and specifies the length of a record in bytes. The length of a record can be determined using the Len function.

Closing a file is performed with the FileClose statement:

CloseFile= FileClose "(" [<FileNumber> {, <FileNumber>}] ")"

If the FileClose statement is used without a FileNumber then all files which were opened with a FileOpen statement will be closed.

For random access files writing and reading data to and from a file is accomplished with the FilePut and FileGet statements:

WriteToFile = FilePut "(" <FileNumber>, <RecordVariable>, [<RecNumber>] ")"

ReadFromFile = FileGet "(" <FileNumber>, <RecordVariable>, [<RecNumber>] ")"

If RecNumber is omitted then the next record is used each time a FilePut or Fileget is executed. The first record in a file is RecNumber 1.

For sequential files, writing and reading data to and from a file is accomplished with the Write and Input statements. In Visual Basic each field of a record must be written to and read from sequential files explicitly. This was not the case for random files where a complete record is read/written as a complete unit.

WriteToFile = Write "(" <FileNumber>, <Variable> {,<Variable>} ")"

ReadFromFile =Input "(" <FileNumber>, <Variable> {,<Variable>} ")"

Multiple data items can be written to a sequential file with one Write statement. Similarly, multiple data items can be read from a sequential file with one Input statement. Data is stored in the file with commas between each data item and any strings are enclosed in double quotes.

Let us now implement our StoreCustomers and RetrieveCustomers procedures in Visual Basic .NET. We will use a random access file for this purpose, although a sequential file would have fulfilled our requirements equally well.

```
Public Sub StoreCustomers (FileName As String)
Dim Count As Integer, FileNum As Integer, CusLen As Integer
FileNum = FreeFile()
CusLen = Len(Customers(0))
FileOpen ( FileNum, FileName, OpenMode.Random, , , CusLen )
Count = 0
While Count <= 500
   FilePut ( FileNum, Customers(Count), Count+1 )
   Count = Count + 1
End While
FileClose ( FileNum )
End Sub
```

We use the FreeFile() function to obtain the next available file number. The Len function is used to find the length of a customer record. As all the customer records are the same length we can find the length of any record, in this case we've found the length of the first record. These values are used as parameters in the statements that follow.

The RetrieveCustomers procedure could be implemented as follows:

```
Public Sub RetrieveCustomers (FileName As String)
Dim Count As Integer, FileNum As Integer, CusLen As Integer
FileNum = FreeFile
CusLen = Len(Customers(0))
FileOpen ( FileNum, FileName, OpenMode.Random, , , CusLen )
Count = 0
While NOT EOF( FileNum )
   FileGet ( FileNum, Customers(Count), Count+1 )
   Count=Count+1
End While
FileClose ( FileNum )
End Sub
```

The RetrieveCustomers procedure above is almost identical to the StoreCustomers procedure in terms of logic. The condition to end the loop uses the EOF function; this function detects the end of the file specified by FileNum.

Note: The VB.NET sequential and random access commands described above are similar to those described in the Software Design and Development Stage 6 - Software and Course Specifications document. There are other more efficient file access methods provided within the .NET framework.



GROUP TASK Activity

A problem occurs with both procedures above if the file does not exist. In this case the OPEN statement will create a new empty file. The procedures still terminate correctly but an empty file will remain on the disk. The Visual Basic Kill command is used to remove files. The syntax is Kill "(" <Pathname> ")". Alter the procedures to correct this fault.



Consider the following:

Imagine your Year 12 class are running a raffle to raise funds for your farewell. There are 500 tickets to be sold to students at the school. First prize is an iPod, second prize is a music voucher and there are 20 other prizes of a free Mars bar from the canteen.

As a keen Software Design and Development class, you decide to create a VB.NET program to manage the raffle. Different routines have been allocated to different members of your SDD class. Fred (not his real name) has been given the task of writing the routine that creates the winners. Fred is not known for his honesty but he is certainly a capable programmer.

```
Public Structure TicketType
   Public Surname AS String
   Public FirstName AS String
   Public Year AS Integer
End Structure
Public Tickets(500) AS TicketType
```

Fig 5.4
The Tickets data structure for the Year 12 raffle.

The main data structure is an array of records called Tickets. The details of the Tickets data structure are given in *Fig 5.4.* Fred has developed an algorithm for this task and has just completed coding the algorithm in Visual Basic .NET.

```
Public PrizeWinners(22) As Integer
Public Sub Winners()
   Dim Winner As Integer, WinNum As Integer, CheckCounter As Integer
   Dim rand As New Random
   WinNum = 1
   While WinNum <= 22
      Winner = rand.Next(0, 500)
      PrizeWinners(WinNum) = Winner
      CheckWinner(WinNum, Winner)
      CheckCounter = 1
      While CheckCounter < WinNum
         If PrizeWinners(CheckCounter) = Winner Then
            WinNum = WinNum - 1
            CheckCounter = WinNum
         End If
         CheckCounter = CheckCounter + 1
      End While
      WinNum = WinNum + 1
   End While
End Sub
```

Fig 5.5
Fred's Winners VB.NET procedure



GROUP TASK Investigation

Examine the Winners procedure ignoring the call to the CheckWinner procedure. Explain in words how Fred has completed the required task.

```
Public Sub CheckWinner(ByRef A As Integer, ByRef B As Integer)
    Dim rand As New Random
    If Tickets(B).Year < 11 And rand.Next(0, 5) <> 0 Then
        A = A - 1
    ElseIf A < 3 And A + Tickets(B).Year <> 13 Then
        A = A - 1
    End If
End Sub
```

Fig 5.6
Fred's CheckWinner VB.NET procedure



GROUP TASK Investigation
Now include the call to the CheckWinner procedure in your investigation. What has Fred done to alter the outcome of the raffle? What is the significance of Fred's use of ByRef rather than ByVal for the parameters to the procedure?



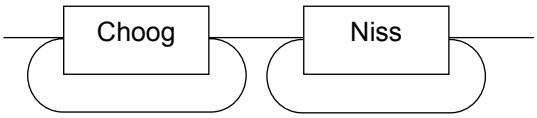
HSC style questions:

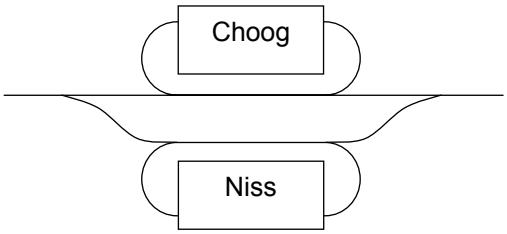
Consider the following EBNF statements when answering Questions 1 and 2.

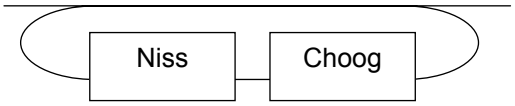
```
Choog = A|B|C|D
Niss = a|b|c|d
Ving = {<Choog>} | {<Niss>}
Ruke = <Choog> <Niss> <Ving>
```

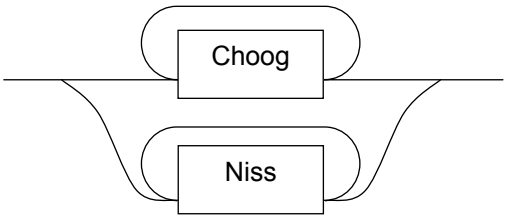
1. Which of the following is a legitimate Ruke?
(A) ABab
(B) AaAaAa
(C) bBccc
(D) CcABCD

2. Which of the following railroad diagrams correctly defines a Ving?

(A)


(B)


(C)


(D)


3. In EBNF a Drim is defined as follows:

Drim = (<Yot> | (<Hok> { <Hok> })) <Yot>

Which of the following best describes a Drim?

(A) A Drim begins with a Yot followed by one or more Hoks with a Yot at the end.

(B) A Drim begins with a Yot or a Hok, which is followed optionally by another Hok with a Yot at the end.

(C) A Drim is two Yots or a Drim is a series of one or more Hoks followed by a Yot.

(D) A Drim is a series of Hoks surrounded by Yots.

Question 4.

A programming language uses the following syntax:

```
Digit = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
Letter = A | B | C | D | E | F
Variable = (<Letter> | <Digit>) {(<Letter> | <Digit>)}
PRINT = PRINT (<Digit> | <Letter> | <Variable>)
Assignment = LET <Variable> = (<Variable>|<Digit>|<Letter>)
Statement = <PRINT> | <Assignment>
Loop = FOR <Variable> = <Digit> To <Digit> {<Statement>} ENDFOR
```

(a) Draw a railroad diagram to show the syntax of an `Assignment`.

(b) Explain why `LET A = 75` is a valid `Assignment`.

(c) Write program code, using the syntax defined above, which will print the numbers from 1 to 5 inclusive.

Suggested Solutions

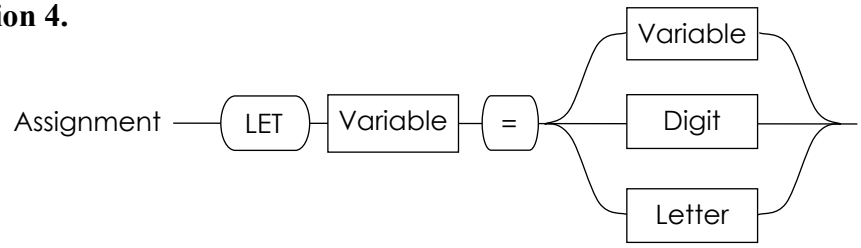
1. (D)

2. (B)

3. (C)

Question 4.

(a)



(b) A is a valid variable because A is a valid letter and a variable can be a single letter.

75 is a valid variable because 7 and 5 are both valid digits and a variable can be one or more digits.

Let A = 75 is a valid assignment because it is in the form LET variable = variable, and A and 75 are both valid variables as explained above.

(c)
```
FOR A = 1 To 5
    PRINT A
ENDFOR
```

SET 5A

1. A metalanguage is:
   (A) either an EBNF or railroad diagram.
   (B) used to make sense of programming languages.
   (C) another name for a high-level language.
   (D) used to describe the syntax of another language.

2. In EBNF {<thing>} means that <thing>:
   (A) is repeated one or more times.
   (B) is repeated zero or more times.
   (C) is optional.
   (D) can take on more than one value.

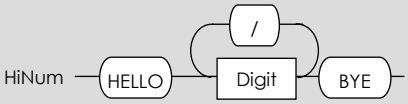
3. Which of the following is true of multi-dimensional arrays?
   (A) Data is stored in records.
   (B) Each data item takes the same value.
   (C) All data items are of the same data type.
   (D) No two data items can have the same value.

4. Declaring a user defined variable involves:
   (A) Defining the data type then declaring the variable.
   (B) Declaring the variable then defining the data type.
   (C) Determining the number of indexes then declaring the variable.
   (D) Opening the data type and then reading its contents.

5. If RND generates a random number in the range 0 to less than 1. INT(RND*6)+4 would result in integers in the range:
   (A) 4 to 10.
   (B) 6 to 10.
   (C) 4 to 9.
   (D) 3 to 9.

6. A multi-dimensional array requires 3 indexes and needs to store 1000 data items. Which Visual Basic statement achieves this task?
   (A) DIM Test(10,10,10)
   (B) DIM Test(9,9,9)
   (C) DIM Test(999,999,999)
   (D) DIM Test(400,300,300)

7. A data structure containing a series of records could be:
   (A) an array of records.
   (B) a file.
   (C) a multi-dimensional array.
   (D) Both A and B.

8. 

   If a digit is defined in EBNF to be 1|2|3|4|5. A syntactically correct HiNum is:
   (A) HELLO/1/2/1/BYE
   (B) HELLO1232/BYE
   (C) HELLO1/2/3/4BYE
   (D) HELLO1/2/3/BYE

9. A data structure is declared in Pascal using the following statements:

```
TYPE
  ThisType = RECORD
       This : Integer;
       That : Boolean
  END;
  ThisThat=ARRAY[1..3] OF ThisType;
```

   Which of the following is correct?
   (A) An array of records exists called ThisThat. It contains six data items.
   (B) Each record of type ThisThat will contain six data items.
   (C) A variable declared with type ThisThat will be an array with 3 records.
   (D) A variable of type ThisThat will contain two records. Each record containing three fields.

10. What is the purpose of the following fragment of Visual Basic code:

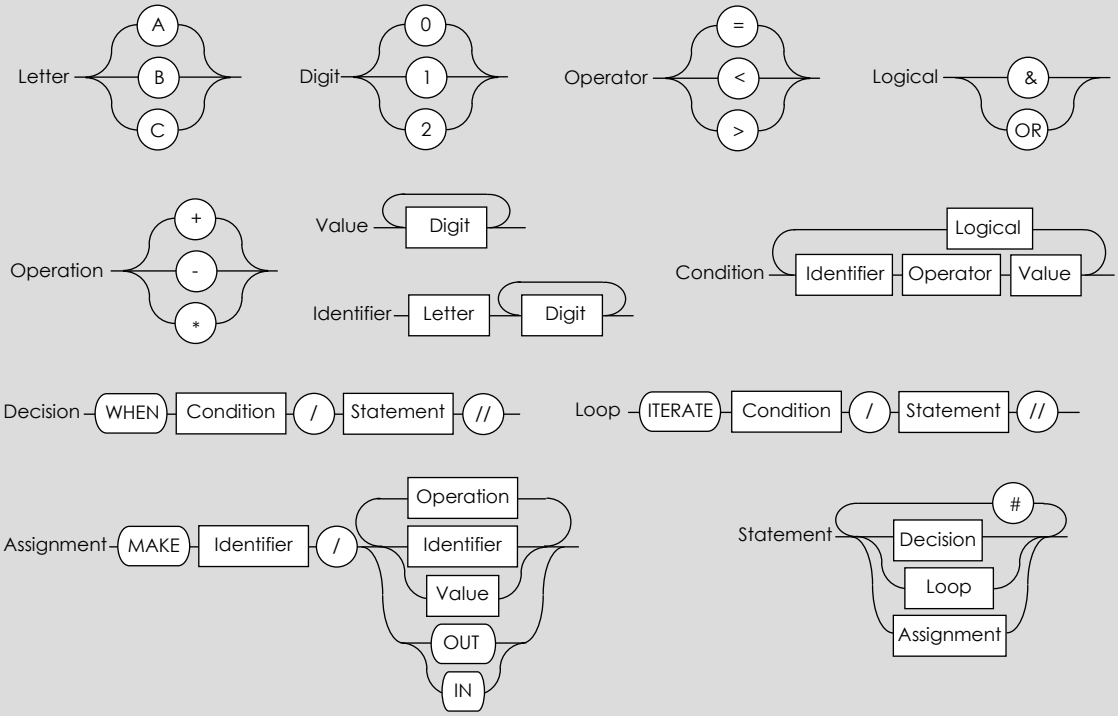
```
Count=1
While Count<=1000
   Result = rand.Next(0, 2)
   Total(Result) = Total(Result) + 1
   Count = Count + 1
End While
```

    (A) A loop counts from 1 to 1000. 1000 random numbers are totalled. Each number is either a 0 or a 1.
    (B) The number of 0s and the number of 1s generated is calculated for 1000 random numbers. Each random number can be either a 0 or a 1.
    (C) One thousand 0s and 1s are generated. The total of all the 0s is stored in Total(0) and the total of all the 1s in Total(1).
    (D) 1000 random numbers from 0 to 1 are generated. Each number is added to the total after it is generated.

11. The ability to store and retrieve data from secondary storage devices is vital to most applications. Explain the processes involved when writing source code to read from and write to disk files.

A new programming language called Zenith has been developed. Zenith uses a Base 3 number system for all its calculations, hence only the digits 0, 1 and 2 are required. In Base 3, we count 1, 2, 10, 11, 12, 100, 101, 102, 110, 111, 112, 120, 121…

The railroad diagrams below describe the syntax of this language:



12. Write down five legal Zenith identifiers.

13. Convert the Zenith assignment command into an EBNF diagram.

14. Zenith uses a multi-purpose assignment command. This command performs the usual assignment function together with input and output. Create a code fragment, in Zenith, that will get a number then print out all the numbers from 1 to that number.

15. Division is not included as a standard operation in Zenith. Write a code fragment in Zenith that performs division using repeated subtraction. You will need two outputs; the quotient and the remainder.

TRANSLATION METHODS IN SOFTWARE SOLUTIONS

Translation is the process of converting high-level code or source code into machine executable code. High-level languages cannot be understood directly by computers, they need to be converted to a form that is understood by the processor. This is similar to speaking in German to an English speaker. The German needs to be translated into English before the meaning can be understood.

Source code is said to be machine independent. This means it can be used on a number of different processors. In reality, some changes are normally required for this to occur. Executable code, on the other hand, is very processor specific. Each family of processors will have different machine language instructions. Source code needs to be translated for use on specific processors. If our German speaker wished to convey the same message to both an English speaker and an Italian speaker he would require two translators.

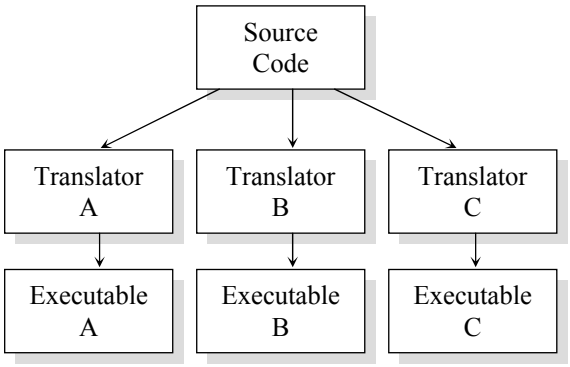


Fig 5.7
Different translators are used to create executable files for different processors using the same or similar source code.

The introduction of the World Wide Web has resulted in languages that can be implemented on a wide range of unknown processors. Languages such as Java, undergo a two-stage translation process. The source code is translated into byte code. This byte code is distributed and translated on the end-user's machine into executable code specific to the user's machine. Each end user machine has translation software installed to perform the byte code to machine code translation, in the case of Java this software is known as the Java Virtual Machine (JVM). A different JVM is installed for different operating systems and devices. This system can be likened to the symbols used to represent mathematical processes. A German mathematician writes his formulas using mathematical symbols, which are understood throughout the world. English and Italian speakers are able to translate these mathematical symbols themselves into their own language.

Let us now examine the two most common methods of translation:

- Interpretation
- Compilation

We will then examine the translation process in detail.



GROUP TASK Discussion
'A large variety of translators are available for the C++ language. Wouldn't it be better if we could just have one standard translator that all C++ programmers could use?'
Do you agree with this statement? Discuss.



GROUP TASK Research
Java executes on a broad range of devices. Using the internet or otherwise create a list of different devices which utilise a JVM to execute Java code.

Interpretation

Each line or statement of source code is translated into machine code and then immediately executed. If errors exist in the source code, they will cause a halt to execution once encountered by the interpreter. The actions of an interpreter are similar to that of a speech interpreter. If our German speaker wishes to have a conversation with an English speaker then the speech interpreter translates each sentence into English as soon as it is spoken in German. The English version is then spoken to the English-speaking recipient. If the speech interpreter cannot understand a German sentence, he is unable to translate it into English. The German sentence must be changed to a form the speech interpreter can understand before the correct translation can take place. Because the interpreter is there at the time the communication takes place, it is easy to make an on the spot change and then continue with the translation process. High-level language interpreters provide a similar advantage to the developer. Errors are generated as they occur and can be corrected. Translation and execution can then continue.

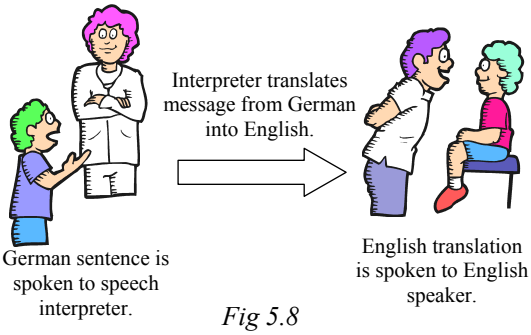


Fig 5.8
Speech translation can be likened to source code interpretation.

Interpretation does slow down the process of execution significantly. Each statement must be translated into executable code before it is executed. The translation process creates a significant time overhead. Because the interpreter only receives one high-level language statement at a time, it is not able to look ahead and translate each statement in the most efficient manner. Other methods of translation are able to optimise their output based on the overall view of the total source code.

Users of interpreted programs must have a copy of the interpreter installed on their machines for execution to take place. This requirement involves further costs and memory overheads for the users of interpreted software. Because the actual source code is distributed to users, the developer has limited control over his intellectual rights. Users can easily access the source code and use it for their own purposes.

Currently interpreted languages are routinely used on web servers to write server side scripts – PHP and Microsoft's ASP are common interpreted examples. When customising existing applications often the scripting language is interpreted.

For most compiled languages an interpreter is also available for use as a development tool by software developers. Many software development environments include an interpreter for use during coding which allows the programmer to execute small segments of code quickly as they are coding. The final distributed product is produced using a compiler.



GROUP TASK Activity
Interpreters translate each of line of code and then execute it if no errors are found. Create a flowchart to describe the operation of an interpreter.



GROUP TASK Research
Make a list of programming languages which are generally interpreted at runtime and briefly outline the main use of each language.

Compilation

The source code is translated into executable code. The executable code can later be executed without any need for the translator. Compilation is a batch process. The input to this process is the high-level source code and the output is the executable file or files. Errors encountered during the compilation process are normally relayed as a series of messages to the programmer. All coding errors are reported at the end of an unsuccessful compilation operation.

The operation of a compiler can be likened to a translator translating a book into a foreign language. Imagine a translator is converting a novel written in German into English. The translator will translate the entire text of the novel. Once the manuscript in English has been produced in its entirety it is printed and distributed. It is difficult for readers of the English version to realise the language used in the original German book. Let's take our analogy one step further. The translator will no doubt find various sentences in the original German difficult to translate into English. The translator would make a note of these sentences. At the end of the translation process, these sentences would be analysed and perhaps the original meaning would be acquired from the author. The translator then goes back over the manuscript correcting their translation. This is similar to a compiler listing all the errors found in the source code and the programmer going back and correcting these errors. There is one important difference however. A compiler does not produce any actual machine code until the entire source code is known to be correct in terms of the syntax of the source code.

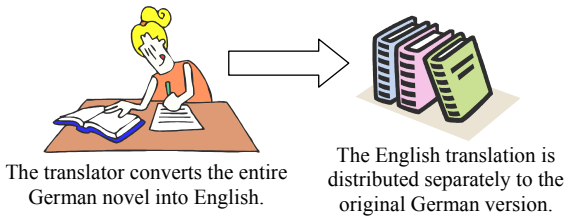


Fig 5.9
Compilation can be likened to the translation of a novel from German into English.

Compilers are used to produce executable code for the majority of commercial applications. Compiled programs generally run faster and more efficiently than similar interpreted products. As a compiled program is made up of machine code instructions, it is very difficult for the details of the original source code to be realised. This protects the intellectual rights of the author of the software. As the source code is not available to users, it is virtually impossible for them to make changes to the product. This makes it easier for software companies to provide quality support for their products, as they know precisely the workings of the application.

Executables created with compilers will always be machine specific. If a product is to operate on a different processor or operating system, then the source code will need to be recompiled using a compiler that produces the appropriate machine code instructions.



GROUP TASK Activity
Compilers translate the entire source code into an executable file or files (assuming no errors are found). These executables are run at some later stage separate to the compiler. Design a flowchart to describe the operation of a compiler.



GROUP TASK Research
Make a list of programming languages which are generally compiled and briefly outline the main use of each language.

The translation process in detail

In this section, we will examine in some detail the processes involved in translating source code into an executable file. We will look at the general processes used by the majority of compilers. Some aspects of this description may not be relevant to some translators. As most programs are compiled rather than interpreted, our emphasis will be on this area.

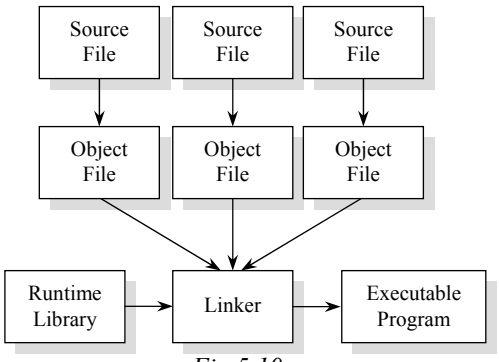


Fig 5.10
The translation process often includes a number of source files together with routines from a runtime library.

The process of compilation involves two distinct steps; the creation of object code from the original source code statements followed by linking this code to existing runtime libraries and dynamic link libraries (DLLs). Large projects will often involve translating a number of source files into object code files. The linker is used to join all the components of a project into one final executable program.

Translating source code into object code is the major task of the compiler. This process always requires the following steps:

- Lexical analysis
- Syntactical analysis
- Code generation

Refer to the DFD in *Fig 5.11* as we consider each of these steps in turn.

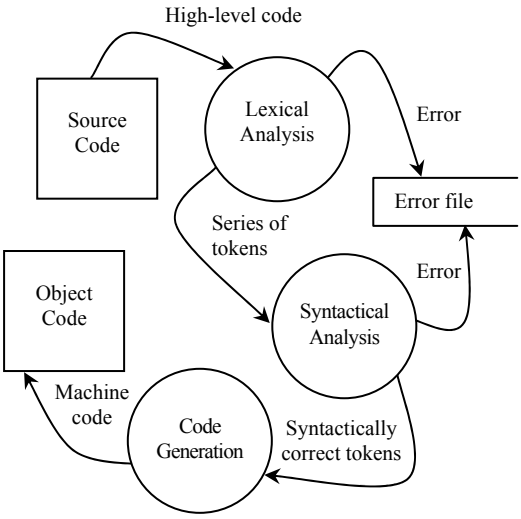


Fig 5.11
Data flow diagram describing the process of translating source code into object code.

Lexical analysis

The word lexical comes from the word lexicon. A lexicon is a wordbook or dictionary containing the vocabulary of a particular language or subject. Lexical analysis is the process of examining each word to ensure it is a valid part of the language. In terms of source code translation, lexical analysis is a process that ensures all reserved words, identifiers, constants and operators are legitimate members of the high-level language's lexicon. At this stage of the translation process we are not interested in whether these language elements are in a sensible order, we just wish to ensure that each is a legitimate language element in its own right.

How is this achieved? A table of symbols or tokens is used. This table will initially contain all the reserved words and operators that are included elements of the language. As the lexical analysis continues, identifiers and constants will be added to the symbol or token table. This table essentially becomes a dictionary or lexicon for the lexical analysis process. Each character in the source code is read sequentially. Once a string of characters, called a lexeme, matches an element in the symbol table, it is replaced by the appropriate token. This process continues until the entire source code has been tokenised. (Comments and white space are ignored during lexical analysis).



Consider the following:

The fragment of Pascal code shown in *Fig 5.12* is being translated by a compiler. Let us examine the process of lexical analysis carried out on this code fragment.

Firstly, let us examine a portion of the symbol table that exists before the lexical analysis commences. For this example, we will use different graphics to represent each token, in reality the tokens would be unique bit patterns.

Token	Lexeme	Description
	PROCEDURE	Reserved word.
	VAR	Reserved word.
	BEGIN	Reserved word.
	END	Reserved word.
	WHILE	Reserved word.
	DO	Reserved word.
	Writeln	Runtime procedure identifier.
	:=	Operator.
	<=	Relational Operator.
	+	Operator.
	(	Symbol.
	)	Symbol.
	;	Symbol.
	:	Symbol.
	Integer	Data type.

```
PROCEDURE OneToTen;
VAR
   Count : Integer;
BEGIN
   Count:=1;
   WHILE Count<=10 DO
      BEGIN
         Writeln (Count);
         Count:= Count+1
      END
END;
```

Fig 5.12
Sample Pascal code fragment.

Each character in the source code is examined in turn. Once a valid lexeme is encountered, the appropriate token is sent on for syntactical analysis. New identifiers and constants are added to the symbol table as they are encountered. The tokenised code fragment after lexical analysis would be:



Items that have been added to the symbol table would be:

Token	Lexeme	Description
❶	OneToTen	Procedure Identifier.
❷	Count	Integer Identifier
❸	1	Numeric Constant
❹	10	Numeric Constant.
❺	1	Numeric Constant.

The lexical analysis process has determined the nature of each element in the source code. Elements that do not exist in the symbol table must either be identifiers or be constants, no other possibilities exist. Notice that each time a constant appears it is assigned its own token. If the same constant is used a number of times, the symbol table will contain an entry for each occurrence. It is better to assign the value of frequently used constants to an identifier and then use the identifier.

Error messages generated as a result of lexical analysis will be in regard to the use of undeclared identifiers, incorrect naming of identifiers or incorrect constant syntax. For example, identifiers encountered during lexical analysis must conform with the language syntax for an identifier. The EBNF production for a Pascal identifier is Identifier=<letter>{<letter> | <digit>}.

Syntactical analysis

The word syntax refers to the patterns or arrangements of words used to form sentences or phrases in a particular language. Syntactical analysis is the process of checking the syntax of sentences and phrases is correct in terms of the grammatical rules of the language. Programming languages have extremely strict syntax rules. Earlier in this chapter we looked at EBNF and railroad diagrams, these are methods of precisely describing the syntax of a programming language. Syntactical analysis is used to check source code statements adhere precisely to these rules. The second part of syntactical analysis involves checking the type of identifiers and constants used in statements are compatible.

Parsing is the process of checking the syntax of a sentence. A parse tree is used to diagrammatically describe the component parts of a syntactically correct sentence. Syntactical analysis makes use of parse trees to check each statement against the precise description of the syntax. The series of tokens delivered for syntactical analysis are constructed into a parse tree based on the rules for the particular language described in EBNF or railroad diagrams. These diagrams provide the essential algorithms for the creation of the parse tree. Each statement in a programming language will have exactly one parse tree, resulting in one precise meaning. As a complete program is made up of a series of statements then only one unique parse tree can be created for a particular program. Spoken language can often be parsed in many ways conveying various meanings. For example, 'Fred put the apple in his lunchbox in his school bag' could mean Fred got the apple out of his lunchbox and put it into his school bag or it could mean Fred put his apple in his lunchbox and then put the lunchbox into his school bag. The syntax of programming languages must allow for only one possible meaning.

If a particular component of the source code cannot be parsed then an error message is generated. The error messages generated as a consequence of parsing will always be in regard to the incorrect arrangement of tokens. In other words, the elements of the statement are not in the correct order stipulated by the language.

Once a parse tree has been created, the data type of identifiers and constants within each statement can be checked for compatibility. This process is known as type checking. For example, an identifier used to store simple integers cannot be assigned an array of characters. Mathematical operators cannot be used with string data. Any errors found during this type checking process are normally reported as type mismatch errors.



GROUP TASK Activity
Create a fragment of code using a programming language with which you are familiar. Create a symbol table and then perform a lexical analysis of your code fragment.



GROUP TASK Discussion
Lexical analysis will not find all the errors in source code. What types of errors will not be identified as a result of lexical analysis? Explain your answer using examples.



Consider the following:

The partial statement WHILE Count<=10 DO has been through the lexical analysis process as described on the previous page. As a result the following sequence of tokens were generated – ✠ ❷ ⌘ ❹ ♓. The syntax of this token sequence is now to be checked.

The token ✠ represents the reserved word WHILE. We examine the railroad diagram for a Pre-test statement and find that the keyword WHILE must be followed by a condition. The next symbol ❷ represents an identifier. We check the railroad diagram for a condition and find that conditions can commence with identifiers. We are now inside the condition railroad diagram and are considering the third symbol ⌘ which is a relational operator. The condition railroad diagram states that a relational operator must follow an identifier so we have no errors so far. The fourth symbol ❹ represents a numeric constant. The condition railroad diagram allows constants to follow relational operators. The condition railroad diagram is now complete so we return to the Pre-test railroad diagram. The fifth token ♓ represents the keyword DO. Upon checking the Pre-test railroad diagram, we find that this is also correct. Hence, our partial statement is syntactically correct.

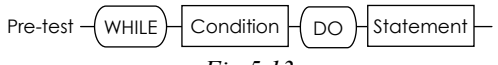


Fig 5.13
Pre-test railroad diagram.

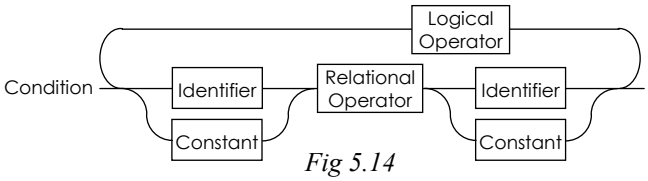


Fig 5.14
Condition railroad diagram.

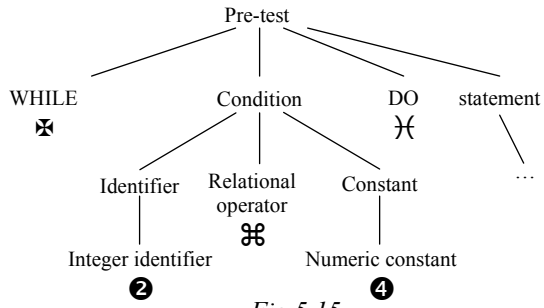


Fig 5.15
Sample Pre-test parse tree.

To complete the syntactical analysis, we examine the data type of identifiers and constants used within statements. In our example, we have only one instance to examine; namely the condition. Since identifier ❷ and constant ❹ are both of the same type, our syntactical analysis is complete.



GROUP TASK Discussion
There is one unique parse tree for each particular piece of source code. Wouldn't it be preferable to have a number of different possible parse trees for each piece of code? Discuss.

Code Generation

If the processes of lexical analysis and syntactical analysis have completed without error then the machine code can be generated. This stage involves converting each token or series of tokens into their respective machine code instructions. Because we know that the tokens are correct and are in the correct order, no errors will be found during code generation.

The parse tree created during syntactical analysis will contain the tokens in their correct order for code generation. The code generation process traverses the parse tree, usually from left to right along the lower branches. Tokens are collected and converted into their respective machine code instructions as the traversal occurs.

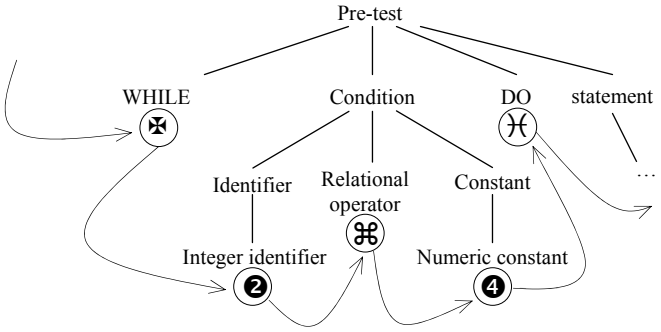


Fig 5.16
Traversing the parse tree collecting tokens.

If a compiler is being used then the resulting machine code instructions are stored in an object file. This file contains all the executable code derived from the source code. Object files do not generally contain links to runtime libraries and dynamic link libraries (DLLs) required for execution. A linker is used to perform this function. Most compilers include a linker as an integral part of the compilation process.



Dynamic Link Library (DLL)
A file containing object code subroutines that add extra functionality to a high-level language. DLLs need to be distributed with applications.

If an interpreter is being used then each instruction once generated will be executed immediately. The interpreter takes care of access to runtime libraries.



GROUP TASK Research
Some development environments require linking to be performed as a distinct task after compilation has taken place. Others automatically perform the functions of a linker. Different methods of optimising the final executable code are also provided. Examine a number of compilers in regard to options available to the developer to change the way the source code is generated.



GROUP TASK Research
Visual Basic .NET and other .NET language compilers generate CIL (Common Intermediate Language) code rather than executable code. CIL code requires the .NET framework to be installed before .NET applications can run. Research to determine why CIL is used by .NET languages rather than generating executable code.



HSC style questions:

Question 1.

Distinguish between the translation of source code into executable code using an interpreter with translation using a compiler.

Question 2.

Explain how a compiler translates source code into executable machine code.

Suggested Solutions

Question 1.

Interpretation is a translation method that converts each line of source code and then executes it immediately. Hence, interpretation occurs at runtime and the interpreter must be present on the user's machine. Compilation translates the entire source code into an executable file. The executable file is then distributed to users. At runtime no translator is needed, therefore compiled programs run faster than similar interpreted programs. In addition, when a program is compiled the users do not have access to the source code, which helps protect the programmer's intellectual property. Some programs are interpreted so that only one version is needed that can execute on different operating systems and CPUs. If the source code is compiled then different executables need to be created for different operating systems and CPUs.

Question 2.

Compilers perform the translation process in three stages – lexical analysis, syntactical analysis and finally code generation.

- Lexical analysis first separates source code statements into their separate words (or lexemes) and then checks that each word is a legitimate part of the programming language – much like checking the spelling of words. Each legitimate part of the language is tokenised. A table specifies precisely what each token represents. Errors are generated as they are encountered. Some compilers halt on the first error whilst others will continue (but not generate code).
- Syntactical analysis is similar to checking the grammar of written sentences. It ensures that the order and arrangement of the tokens complies with the syntax rules of the language, essentially the automated equivalent of using EBNF definitions to check syntax. The definition of a program, statements, conditions, and so on are used to create a parse tree of all the tokens within the program. The parse tree arranges the tokens into a logical unique order. Syntax errors are generated as they are encountered. Some compilers try to continue so that all syntax errors are reported whilst most halt on the first error.
- If both lexical and syntactical analysis complete without error then code generation begins. Code generation converts each token or sequence of tokens into the corresponding machine language statements. The parse tree is traversed along its lower branches and tokens are translated in sequence. No errors are generated during code generation as all tokens are legal and their order is also correct. Finally links are created to any external files required at runtime – such as DLLs and other executable libraries. The result is one or more files containing machine code instructions.

SET 5B

1. The process of converting source code into machine executable code is known as:
   (A) compilation.
   (B) translation.
   (C) interpretation.
   (D) execution.

2. Machine independence means that the source code:
   (A) can be translated for use on a variety of processors.
   (B) contains machine language instructions that are common to all processors.
   (C) is able to be executed on a variety of computer platforms.
   (D) can be understood by humans but computers are unable to execute the code directly.

3. Lines of code are executed as soon as they have been converted into executable code. This occurs during:
   (A) compilation.
   (B) translation.
   (C) interpretation.
   (D) documentation.

4. Executable code will always be machine specific. Why is this the case?
   (A) Machine specific code protects the authorship rights of software developers.
   (B) Different processors have differing machine code instruction sets.
   (C) Each computer can be configured to perform in a number of different ways.
   (D) Executable code is very difficult for humans to read. It is written specifically for computers.

5. When a user is executing code, syntax errors are only possible when using:
   (A) compilation.
   (B) translation.
   (C) interpretation.
   (D) byte code.

6. Execution does not occur during the translation process when using:
   (A) compilation.
   (B) byte code.
   (C) interpretation.
   (D) both A and B.

7. The process of connecting object files to runtime libraries and DLLs is performed by a:
   (A) compiler.
   (B) dynamic link library.
   (C) linker.
   (D) translator.

8. Ensuring each element of the source code is a legitimate part of the language is part of:
   (A) code generation.
   (B) syntactical analysis.
   (C) lexical analysis.
   (D) tokenising.

9. A statement that has been parsed correctly is known to be:
   (A) lexically correct.
   (B) syntactically correct.
   (C) logically correct.
   (D) Both (A) and (B).

10. During the code generation stage of translation:
    (A) no errors are generated.
    (B) syntax errors are generated.
    (C) logical errors will be highlighted.
    (D) an executable file is created.

11. Compare and contrast the functions of an interpreter and a compiler.

12. A large software development company is working on a new product. This product will be used to automate the processes of a chain of stationary suppliers. There are six programmers working on the project. Discuss how both forms of translation could be used to assist in the development of this product.

13. Compilation is the most popular method of translation. Why do you think this is the case? Explain the processes of lexical analysis, syntactical analysis and code generation in regard to compilation.

We examined the imaginary programming language 'Zenith' in Set 5A. Use the railroad diagrams from Set 5A to assist in answering questions 14 and 15.

14. Perform a lexical analysis of the Zenith statement that follows. Create a symbol table and tokenise the statement.

    ITERATE B12 < 102 / MAKE A1 / A1 + 1 # MAKE B12 / IN //

15. Create a parse tree using your results from question 14.

THE ROLE OF MACHINE CODE IN THE EXECUTION OF A PROGRAM

In the previous section, we examined the process of translating high-level code into machine code. In this section, we examine how the CPU executes machine code commands which are the building blocks of all programs.

Two questions fundamental to our discussion require our attention:

- What is machine code?
- How is machine code executed by the CPU?

What is machine code?

Machine code is the native language of the CPU. Machine code is a collection of machine language (ML) instructions. Each instruction is in binary, that is, a string of 1s and 0s. For example, 001001001110111011001011. The CPU contains a series of microcode instructions which are permanent and therefore cannot be modified. A machine language instruction typically translates into one or more microcode instructions. Different CPUs will contain a different microcode instruction set hence they require a different set of machine language instructions.

Strings of binary digits may well make perfect sense to the CPU however for us humans they appear rather daunting. Assembly languages are a way of writing machine code that can be understood by humans. In most cases, one assembly language statement corresponds directly to a single machine language statement.



Consider the following:

Let us consider the fragment of assembler code shown in *Fig 5.17*. This code is written for the Intel 8086 family of processors and will execute on modern Intel CPUs. Essentially, this code waits for the escape key to be pressed and then exits the program.

```
escwait:  mov   al,0
          int   16h
          cmp   al,1bh
          jnz   exit
          jmp   escwait
exit:     mov   ax,3
          int   10h
          mov   ax,4c00h
          int   21h
```

Fig 5.17
Assembler code fragment for the Intel 8086 family.

This fragment contains two procedures, one called escwait and one called exit. Actually, they are not really procedures rather they are labels. Labels in assembler are markers in the code that can be returned to using a jump.

There are many registers within the CPU. The accumulator is a general purpose register used to accept the result of computations performed by the ALU. In 8086 assembler the accumulator is signified by the letters ax, a stands for accumulator and x is used to signify it is a general purpose register. This code will operate on earlier 16 bit processors so each register is 16 bits long. As a consequence, each register can be split into two bytes (two lots of eight bits). In assembler, these parts of the accumulator are named ah meaning accumulator high byte and al meaning accumulator low byte. For example if ah=11000110 and al=00111111 then ax=1100011000111111.



Register
A temporary storage location within the central processing unit. The size of a register is the same as the word size of the processor.

The CPU needs to communicate with outside devices. Communication is accomplished via interrupts. Essentially, an interrupt is a communication channel into and out of the CPU. Each communication channel is assigned a unique value. For example interrupt 00010110 or 16 in hexadecimal (16h) is a channel that receives data from the keyboard and causes it to be placed into the accumulator. Interrupt 00010000 or 10 in hexadecimal (10h) is used for communication with the screen.



Interrupt
In relation to the CPU an interrupt is a call to an outside device. The device responds by reading or writing to the accumulator.



Flag
In relation to the CPU, a flag is a single bit stored within a register. Flags are used to indicate an event has occurred within the CPU.

Let us now examine the assembler code in *Fig 5.17* in detail. Statements are executed by the Arithmetic Logic Unit (ALU) and the result is stored in the accumulator.

escwait:	mov	al,0	Moves the value 0 into the register al. This initialises the al register. (Escwait: is a label, we need to return to this position later).
	int	16h	Interrupt 16h is the keyboard. This statement returns the value of the key pressed to the accumulator. The ASCII code of the key is held in al and a code signifying the particular key to ah. For example typing an "a" results in ah=1eh and al=61h=97, 97 is the ASCII code for "a". Typing an "A" results in ah=1eh and al=41h=65, 65 is the ASCII code for "A".
	cmp	al,1bh	Compares the value in al with the value 1bh. If they are equal then the flag called nz is set to true. 1b in hexadecimal is 27 in decimal, which is the ASCII code for the ESC character generated by pressing escape on the keyboard.
	jnz	exit	J stands for jump, in this case a jump to the statement following the label exit is performed if the flag nz is true. True is represented by a binary 1 and false by a binary 0. The flag nz is a single binary digit.
	jmp	escwait	An unconditional jump to the statement following the label escwait is performed.
exit:	mov	ax,3	The value 3 is moved into the accumulator.
	int	10h	The screen is instructed to read the accumulator value 3. This clears the screen. A nice thing to do before we exit!
	mov	ax,4c00h	Place the value 4c00h into the accumulator.
	int	21h	Interrupt 21h is used to communicate with the operating system. The value 4c00h held in the accumulator causes the program to terminate and hands control back to the operating system.



GROUP TASK Activity
Modify the above assembler program so that it requires the escape key to be pressed twice before the program terminates. Use the register dx that has parts named dh and dl in your solution.



GROUP TASK Discussion
'We have only just scratched the surface of assembler code. Every program ever written could have been written in assembler code.'
Why is this statement true? Discuss.

Let us now consider how assembler code is translated into machine code instructions that the CPU can understand and execute. In the large majority of cases, an individual assembler code statement is translated into an individual binary machine language instruction. Each CPU design will have its own instruction set. An instruction set is the list of microcode procedures permanently stored in the processor. *Fig 5.18* shows the mnemonics associated with each of the microcode procedures available in the Intel 8086 processor instruction set.

aaa	aad	aam
aas	adc	add
and	call	cbw
clc	cld	cli
cmc	cmp	cmps
cmpxchg	cwd	daa
das	dec	div
esc	hlt	idiv
imul	in	inc
int	into	iret/iretd
jxx	jcxz/jecxz	jmp
lahf	lds	lea
les	lock	lods
loop	mov	movs
loope/loopz	loopnz/loopne	mul
neg	nop	not
or	out	pop
popf/popfd	push	rcl
rcr	pushf/pushfd	rep
repe/repz	ret/retf	repne/repnz
rol	ror	sahf
sal/shl	sar	sbb
scas	shl	shr
stc	std	sti
stos	sub	test
wait/fwait	xchg	xlat/xlatb
xor		

Fig 5.18

The Intel 8086 processor instruction set. All these instructions are still used in current Intel processors.

The 8086 assembler instruction adc ax,2dh adds the value 2dh to the value contained in the accumulator. The carry flag, cf, is set if the result of the addition overflows the word size of the processor. The hexadecimal machine language equivalent is 15 2D. In binary, this statement becomes 00010101 00101110. In other words, the instruction adc ax is permanently stored in the processor and is accessed using the code 15h. The microcode instructions contained within the CPU explain how to execute the command 00010101 00101110. Execution of machine language commands may take a number of microcode steps. The next section describes how each microcode step is executed by the CPU.



GROUP TASK Discussion

Let us assume the accumulator register ax contains 10110111. The machine language instruction adc ax,e2h is now executed. What are the binary values held in the accumulator ax and the carry flag cf?

How is machine code executed by the CPU?

As previously described, the CPU contains permanent instructions that describe how each machine language instruction is to be executed. These instructions are known as microcode or firmware. Each machine language instruction activates a microcode procedure requiring execution of one or more microcode instructions. Each microcode instruction is executed using the fetch-execute cycle and the integrated circuits within the CPU.

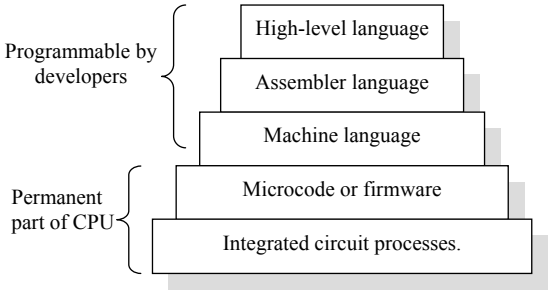


Fig 5.19

Different levels of code from high-level languages to processes performed by the integrated circuits within the CPU.

Let us now examine how the CPU executes each inbuilt microcode instruction. To do this requires some knowledge of the internal structure of the CPU. Different models of CPU will vary, however all contain three basic units; the control unit (CU), the arithmetic logic unit (ALU) and registers. Each of these components is made up of a combination of circuits. Communication between these three components is via a series of connections. Each connection is called a bus.

Fig 5.20 shows a simplified view of a typical CPU. Five registers are shown on this model. The accumulator (ACC), the instruction register (IR), the program counter (PC) and two other registers labelled X and Y. Remember, a register is merely a circuit that is able to store data. Control bus connections are shown on the model using thin lines, whilst data and address bus connections are shown using thick lines.

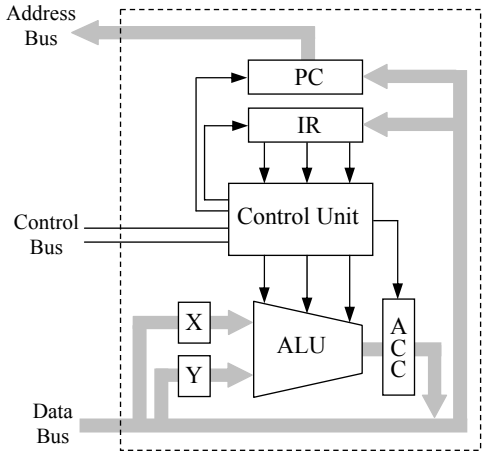


Fig 5.20
A simplified model of the CPU

A complete fetch-execute cycle is required to execute each microcode instruction. The following steps form this cycle:

1. Fetch instruction from primary memory and place in the instruction register.
2. Set the program counter to the next instruction's address. As instructions are stored sequentially this step merely increments the program counter by 1.
3. Decode the instruction. This is performed by the control unit. Involves determining the nature of the instruction.
4. Load operands from memory (if required) into a register. For example, an add instruction will include an operand that is a number. A store instruction will include an operand that is a memory address. Some instructions, such as increment, have no operands.
5. Execute the instruction and store the result. The ALU usually performs this task. Normally the result is stored in the accumulator. Some tasks require more than one result so other locations are used. For example, an add with carry will store the main result in the accumulator and set the carry flag appropriately.
6. Reset the fetch-execute counter back to one ready to commence the process again from step 1.

In the Preliminary course, we discussed the fetch-execute cycle in terms of 4 steps; fetch, decode, execute and store. We have added steps 2, 4 and 6 to our Preliminary course explanation. Step 5 combines the execute and store steps as both these are performed by the ALU after one signal from the control unit. The control unit sends a signal to the appropriate component via the control bus to begin each of these 6 steps. For example, in step 5 the control unit sends a signal to the ALU indicating which microcode instruction is to be executed. The control unit sends one signal for every tick of the system clock. If the processor is operating at 2GHz this means approximately 2 billion ticks of the system clock will occur every second.



Consider the following:

The 8086 microcode instruction `adc ax,2dh` is the next instruction to be executed. This instruction is held in memory location 8. Let us examine the steps of the fetch-execute cycle as this instruction is processed. To simplify matters we will use mnemonics for instructions and hexadecimal values rather than binary.

PC	IR	ACC	X	Y	Fetch-execute counter	Explanation
8	mov ax,1fh	1fh	1bh	33h	0	The initial values held in the registers after the previous fetch-execute cycle finished. The last instruction executed is held in IR.
8	adc ax,2dh	1fh	1bh	33h	1	Increment fetch-execute counter. The *load IR from memory* control signal is sent from the control unit. The program counter holds a value of 8, so the contents of primary memory location 8 is fetched and placed into the instruction register.
9	adc ax,2dh	1fh	1bh	33h	2	Increment fetch-execute counter. The *increment PC register* signal is sent from the control unit. The program counter is incremented ready for the next instruction.
9	adc ax,2dh	1fh	1bh	33h	3	Increment fetch-execute counter. The control unit decodes the instruction. It now knows that there are two operands required for execution and that the ALU is required for execution of the statement.
9	adc ax,2dh	1fh	1fh	2dh	4	Increment fetch-execute counter. The value in the accumulator is copied to the X register. The value 2dh is placed in the Y register.
9	adc ax,2dh	8ch	1fh	2dh	5	Increment fetch-execute counter. The *execute adc* signal is sent to the ALU by the control unit. The ALU adds the values in X and Y. The result of the addition is stored in the accumulator. The carry flag is set to zero as there is no carry.
9	adc ax,2dh	8ch	1fh	2dh	0	Increment fetch-execute counter. The fetch-execute counter momentarily has a value of 6. The fetch-execute cycle is over so the fetch-execute counter is set to zero ready for the next cycle to commence.



GROUP TASK Discussion

The program counter register (PC) keeps track of the memory address of the next instruction. Discuss how this register can be used to cause a loop to be implemented.



GROUP TASK Discussion

The microcode instruction `jmp e2h` causes a jump to the statement stored in memory location e2h. Describe the steps of the fetch-execute cycle in terms of the execution of this statement.

Note: Some details have been omitted in the above explanation for purposes of clarity. In actuality there are further registers used during the fetch-execute cycle, including a number of flags. The microcode instruction used is not a precise 8086 instruction.



HSC style questions:

1. In regard to source and machine language code, which of the following is true?
- (A) Source code includes instructions written in a particular programming language. Machine language code includes instructions written for a particular type of CPU.
- (B) Source code includes instructions written for a particular type of CPU. Machine language code includes instructions written in a particular programming language.
- (C) Source code is compiled programming language code. Machine language code is specific to a particular type of CPU.
- (D) Source code includes instructions written in a particular programming language. Machine language code uses short mnemonics that correspond to one or two CPU instructions.

2. In the CPU, where is the address of the next instruction to be executed held?
- (A) Instruction register
- (B) Accumulator
- (C) Program counter
- (D) Arithmetic logic unit

3. Prior to execution, each machine code instruction and required data is fetched from primary memory. Where are they stored during execution?
- (A) Registers
- (B) Control unit
- (C) RAM
- (D) Program counter

4. Which of the following is true in regard to calling functions within dynamic link libraries (DLLs) from within software applications?
- (A) The functions within the DLL are compiled with, and form part of the executable code.
- (B) The DLL file forms part of the source code for the application.
- (C) The DLL file must be distributed and installed along with the application.
- (D) The DLL file must be present on the user's machine at runtime.

Question 5.

The following assembler code is for an 8085 processor. LDA <addr> loads the accumulator (register A) with data stored at <addr> and STA <addr> stores the contents of the accumulator at <addr>. B is another register in the 8085.

```
LDA 8510
MOV B, A
LDA 8511
ADD B
STA 8512
HLT
```

(a) What is the purpose of the above assembler code?

(b) Describe the processes performed within the 8085 CPU as it executes instructions in the above assembler code.

Suggested Solutions

1. (A) Source code is the language of the high level language whilst machine code is the binary language of the CPU.

2. (C) The program counter is incremented so that it points to the address of the next instruction. To implement a loop a machine language instruction is used to alter the address in the program counter.

3. (A) During execution the instruction is held in the instruction register which is simply a register within the CPU.

4. (D) A DLL is a file which contains a library of routines which a program is able to call. A particular DLL file may already exist on the user's machine or it may be installed with the application, however it must be present for the application to run.

Question 5.

(a) The assembly code loads two values: from memory address 8510 and memory address 8511. These are added together and the result is stored in memory address 8512.

(b) When each of the instructions is executed the fetch-execute cycle operates and follows a series of steps:

- Initially, the program counter is set to the address in memory of the first instruction in the program.
- The instruction is fetched from primary memory into the instruction register. For example, the LDA instruction is fetched and placed into the instruction register.
- The program counter is incremented so that it points to the next instruction.
- The instruction is decoded by the control unit so that the nature and purpose of the instruction can be determined. For example, the LDA instruction is decoded and determined that it needs to load the accumulator with the data from a particular memory address.
- Any operands are loaded from memory. For example, the LDA instruction operand is the memory address 8510 so data from that address is loaded into a register.
- The ALU executes the instruction and the result is stored in the accumulator. For the LDA instruction the data in address 8510 is copied into the accumulator.
- The fetch-execute cycle repeats for the next instruction.

SET 5C

1. Machine language is:
   (A) assembly code instructions.
   (B) hard wired into the CPU.
   (C) binary code used to initiate microcode instructions.
   (D) the same for all CPU designs.
2. Assembly languages:
   (A) are used to make machine language more understandable for humans.
   (B) use mnemonics which correspond to machine language instructions.
   (C) are low level languages.
   (D) All of the above.
3. The fetch-execute cycle occurs:
   (A) six times for each microcode instruction.
   (B) once for each microcode instruction.
   (C) for every tick of the system clock.
   (D) once for each high-level language instruction.
4. The word size of a processor determines the:
   (A) length of each microcode statement.
   (B) number of statements that can be executed at the same time.
   (C) number of bits that are processed at any one time.
   (D) time taken to perform a microcode instruction.
5. The main purpose of the accumulator is to:
   (A) store results of processing by the ALU.
   (B) store values about to be processed by the ALU.
   (C) perform addition.
   (D) maintain a record of the memory location of the next instruction.
6. Repetition is achieved in machine code by:
   (A) using a pre-test loop.
   (B) resetting the contents of the accumulator.
   (C) causing the control unit to signal the start and end of a loop.
   (D) setting the program counter to the location of the start of a loop.
7. A temporary storage location within the CPU is called a:
   (A) register.
   (B) accumulator.
   (C) interrupt.
   (D) flag.
8. A single bit within a register that is used to indicate that an event has occurred is called:
   (A) a flag.
   (B) a register.
   (C) a microcode instruction.
   (D) an interrupt.
9. Assembly language interrupt statements are used to:
   (A) read or write to the accumulator from an outside device.
   (B) cause execution to pause whilst data is received.
   (C) ensure data read from outside devices is correct.
   (D) differentiate between different devices.
10. The control unit's primary function is to:
    (A) make sure instructions are executed in the correct sequence.
    (B) ensure the fetch-execute cycle continues without error.
    (C) direct the operation of other CPU components.
    (D) perform mathematical calculations and comparisons.

11. Write a series of 8086 assembler instructions to perform the following:
    - Each keystroke is read in turn from the keyboard.
    - If two consecutive keystrokes are the same then the program exits.
12. The assembler statement `cmp ax, 1` compares the value currently in the accumulator with the value 1. If they are equal then the `nz` flag is set to true. Describe the sequence of events that occur during the fetch-execute cycle to execute this statement.
13. The control unit is central to the operation of the CPU. Describe the control unit's role in the execution of machine code statements.
14. The CPU contains three buses; the control bus, the address bus and the data bus. What is a bus? What is the purpose of each bus in terms of the fetch-execute cycle?
15. Differentiate between high-level languages, assembler languages, machine language and microcode. Give an example of a typical instruction in each language.

TECHNIQUES USED IN DEVELOPING WELL-WRITTEN CODE

In this section, we introduce a number of techniques and tools available to the software developer to assist in the task of code creation. These techniques and tools aim to reduce the number of errors in the final product and improve the maintainability of the software over time.

Developing well-structured and elegant solutions is the most important technique for reducing the number of errors encountered during coding. Algorithms that are well written and modularised ease the coding and debugging process.

Before we discuss techniques and tools for error detection and correction, we need to examine the types of errors that can occur. We look at syntax, logic and runtime errors; what causes these errors and how the programmer discovers them.

We then examine a number of techniques and software tools that can be utilised to test code, and detect and identify errors. Once an error has been precisely detected and identified, correction is normally a formality. The identification of the error in most cases implies a method of correction.

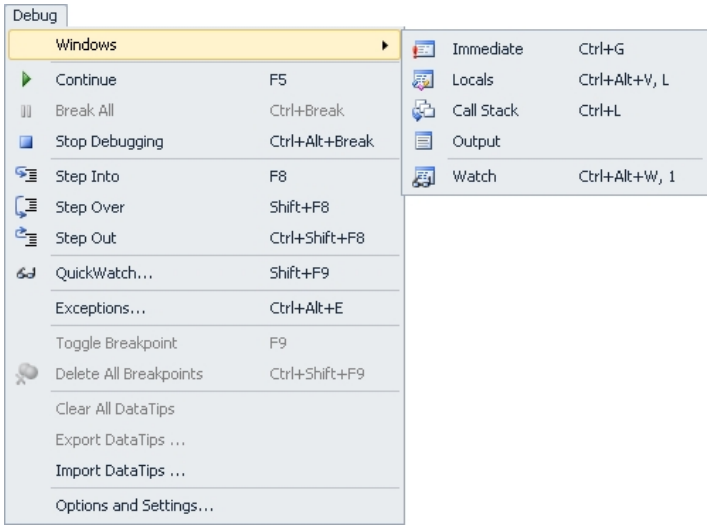


Fig 5.21
Debugging tools available in VB.NET.

Error detection/correction techniques

- Stubs
- Drivers
- Flags
- Debugging output statements
- Peer checking
- Desk checking
- Structured walkthrough
- Use of expected output

Software debugging tools

- Breakpoints
- Resetting variable contents
- Program traces
- Single line stepping
- Watch expressions



GROUP TASK Investigation
Examine a number of programming language environments. Make a list of all the software tools available in each environment for debugging code.



GROUP TASK Discussion
'Well designed algorithms will result in error free code.' Do you agree with this statement? Explain your response with the use of examples.

THE USE OF GOOD PROGRAMMING PRACTICES

Planning, planning and more planning this is a sure method of ensuring well thought out and well structured programs. In chapter 3, we examined a number of methods of modelling a software system. These models, particularly structure diagrams and dataflow diagrams, describe the structure of the final product. In chapter 4, we examined the creation of algorithms from these system models. Algorithms that are clear and accurate greatly assist in the process of generating error free code.

Good programming practice means using a clear modular design where each subroutine performs one logical task. Solutions need to be efficient and of high quality. Error free code is more likely to be produced under these circumstances. We also need to consider the needs of future maintenance programmers who will be altering the code.

Use of a clear modular structure

Software projects should be developed using a clear modular approach. This refers to the top-down design of the project. Structure charts can be used to clearly describe the top-down design. It is important for programmers to adhere to the modular structure indicated on system models and in algorithms. As a result, the identification of the module where an error has occurred is simplified.

In general, the mainline (main program) should be clear and uncluttered so it is easy to follow and identify calls to subroutines. Each subroutine in a well-structured program can be coded independently. The subroutine can be tested and any errors encountered corrected. The subroutine is only included in the project once it has been rigorously tested on its own. Once included in the project the subroutine is tested in conjunction with routines above and below it in the top-down design. This system of subroutine creation and testing ensures quality code that is reusable in future projects.



Consider the following:

The structure chart in *Fig 5.22* describes the subroutines of a cash register program. Let us consider the coding of the TallyItems subroutine. This routine can be developed and tested independently of the other subroutines. Two small routines, called stubs, would need to be written to take the place of the ScanItem and ReduceStock routines. These stubs merely return any required values to the TallyItems routine when called. In this way, the routine can be thoroughly tested and debugged without being affected by possible errors in other routines. Later, the completed TallyItems routine is added to the project. The routine is then tested in conjunction with routines above and below.

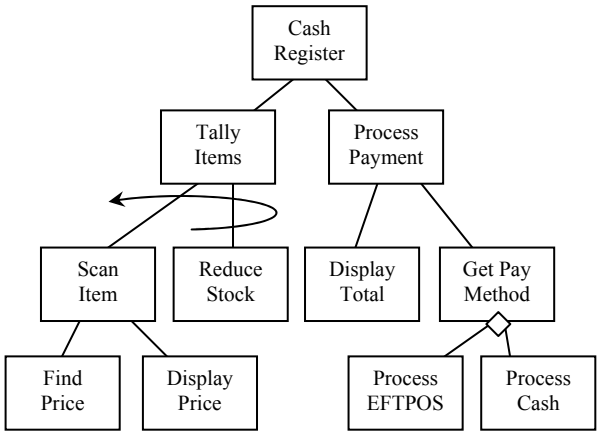


Fig 5.22
Structure chart for a cash register program.
(Shown without parameters)



GROUP TASK Discussion
Examine the cash register structure chart in *Fig 5.22*. Add parameters to this chart appropriately. How does the passing of parameters assist programmers to test subroutines independent of the main project?

One logical task per subroutine

What do we mean by a logical task? This depends on the level of the subroutine within the overall top-down design. A good way of ensuring each subroutine contains only one logical task is to examine the name of the routine. A short succinct name should be able to describe the processing occurring within the subroutine. The algorithm for the subroutine should fit comfortably on a single page. When algorithms grow to become longer than a page then they become difficult to understand and hence difficult to debug and maintain.

Testing a project where the task performed by each subroutine is clear is a simpler process. Errors often occur that can be linked to a particular task. If the identified task is contained to a particular subroutine, which deals solely with that task, then isolating the source of the error is that much easier.

Restricting subroutines to one logical task enhances the reusability of the code. For example, in the cash register structure chart in *Fig 5.22* the ProcessEFTPOS subroutine could easily be used again in future projects requiring EFTPOS transactions.

Quality of solution

A quality solution is a solution that is superior in performance, design and maintainability to other solutions. Quality assurance strategies ensure and confirm that quality solutions will be understandable yet will complete the required task more efficiently than other solutions. The most efficient solutions often involve deep insight and intense problem-solving abilities. Often the most efficient solutions involve generalisations and lateral thinking. Lateral thinking involves making associations that do not follow a particular train of thought.



Consider the following:

The ProcessCash subroutine in the cash register project (see *Fig 5.22*) is being coded. A function, called from this routine, is to round a currency value to the nearest 5 cents.

To complete this task we could extract the last cents digit. If the last digit is a 1 or a 6 we subtract 1 cent, 2 or 7 we subtract 2 cents, 3 or 8 we add 2 cents, 4 or 9 we add 1 cent. *Fig 5.23* shows this function coded in Visual Basic. This solution will of course work, however it is not particularly efficient and contains a variety of quality issues.

This problem can be solved in one line; find the integer part of 20 times the currency value plus 0.5. Divide this answer by 20 and the job is done. The solution implemented in Visual Basic, is shown in *Fig 5.24*. This routine is also a more general solution. The value 20 is the number of times 5 goes into 100. This strategy could be used to write a more general rounding subroutine.

```
Public Function RoundTo5 (CurrVal As
Decimal) As Decimal

Dim LastDigit, TempCurrVal, Final as Decimal

'Ensure CurrVal has 2 dec places
TempCurrVal = Int(CurrVal * 100 + 0.5) / 100

'Extract last cents digit
LastDigit = Int(TempCurrVal * 10)
LastDigit = TempCurrVal * 10 - LastDigit
LastDigit = LastDigit * 10

'Round to nearest 5 cents
Select Case LastDigit
   Case 1, 6: Final = TempCurrVal - 0.01
   Case 2, 7: Final = TempCurrVal - 0.02
   Case 3, 8: Final = TempCurrVal + 0.02
   Case 4, 9: Final = TempCurrVal + 0.01
   Case Else: Final = CurrVal
End Select

Return Final

End Function
```

Fig 5.23
Initial RoundTo5 function in Visual Basic.

```
Public Function RoundTo5 (CurrVal As
Decimal) As Decimal

Dim Final as Decimal

'Round to nearest 5 cents
Final = Int(20 * CurrVal + 0.5) / 20

Return Final

End Function
```

Fig 5.24
Efficient RoundTo5 function in Visual Basic.

Appropriate use of control structures and data structures

Control structures, in particular selection and repetition should only use recognised statements such as pre-test and post-test loops. The condition for the loop should determine the only exit from the loop. There should not be unconditional jumps which cause control to exit a loop or selection unexpectedly. The use pseudocode when writing algorithms ensures only correct control structures can be used, whereas flowcharts allow flow lines to be drawn in non-standard ways.

Care is required when designing and coding repetition structures. Programmers should consider unusual cases such as reading an empty file, processing (or not processing) sentinel values and reacting to unexpected inputs so that infinite loops or other errors do not occur. The data type of variables should also be considered when designing terminating conditions. If a loop should end once a counter exceeds a value it is better to use "greater than" or "less than" operators rather than "equals" as some floating point values cannot be represented exactly and hence an "equals" condition may never be met.

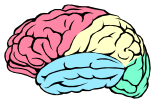
Data structures such as arrays, records, arrays of records and others should be selected and designed to assist the processing to be performed. For example, if a program stores and processes customer details including first name, surname, phone and email then the programmer could use four simple parallel arrays for each of these fields or they could define a record structure and then an array of these records. If four separate arrays are used then any reordering of customers requires the programmer to ensure all the arrays are reordered correctly. If an array of records is used then data about an individual customer remains together as complete records are reordered rather than fields.

Writing for subsequent maintenance

Coding should be done in such a way that future maintenance programmers can easily update the code as new requirements come to light. All the points mentioned above will help to simplify the tasks of maintenance programmers.

Clear documentation within the code, such as comments, appropriate identifier names and indenting within control structures will make understanding the logic of the code clear. Later in this chapter, we examine a number of forms of documentation that will assist the task of maintenance programmers.

If a constant value is to be used throughout the code, it should be assigned to an identifier. The identifier is used rather than the actual value. Maintenance programmers need only change the value once.



Consider the following:

The government decides to progressively remove five, ten and finally twenty-cent pieces from circulation. Cash transactions will now need to be rounded appropriately as each coin is withdrawn from circulation.



GROUP TASK Discussion

Examine the RoundTo5 function in *Fig 5.24*. Develop a function that could be used to round a currency value to the nearest 5 cents, 10 cents, 20 cents or 50 cents. A call to this function should be of the form RoundTo(CurrVal, RoundVal) where RoundVal can take the values 5, 10, 20 or 50.

Version control and regular backup

During coding all programmers should regularly save their work. This is to prevent loss of the source code should the power go out however it also allows different versions of the program or more often the current module or even subroutine to be stored. Often when coding programmers arrive at a solution but then wish to test various modifications. By implementing a system of version control it is possible to efficiently maintain different versions of particular modules. If a modification does not improve the solution then the programmer can revert to an earlier version.

Version control also allows complete applications to be continuously developed with regular versions being released to users. The version control system (VCS) maintains copies and information in regard to all the modules and subroutines required to build a specific version of the final application. This means it is possible to compile any previous version of the application at any time in the future.

Regular backup copies should be made of all source code to prevent against loss. It is important to make copies to different physical devices to prevent against hardware failure and also to make copies to different locations in case of fire or other disasters. Commonly backups are made on at least a daily basis with offsite storage on at least a weekly basis. Online internet based storage is becoming popular for backups as internet speeds continue to increase. Online storage allows regular backups to offsite storage to be performed automatically.

Version control systems (VCS) are routinely used by teams of programmers. A VCS performs all of the above plus it allows multiple developers to work on the same project. Commonly a central repository contains all the files and individual developers must check out the file they wish to modify. The system maintains a record of every change made by every developer so that it is possible to track all changes and also to revert to previous versions of any file as they appeared at any point in time.



GROUP TASK Research
Research to find common version control systems (VCS). Identify the main features available within VCS products.



GROUP TASK Discussion
Is version control only relevant to source code development? Discuss.

Recognition of relevant social and ethical issues

Social and ethical issues that should be considered when writing source code include:

- When using code written by others it is important to respect their intellectual property rights including but not limited to their legal rights.
- Thorough testing to ensure software performs correctly and as expected.
- Documentation for future maintainers of the software. Many successful applications remain in use for decades. It is important that the source code can continue to be understood.



GROUP TASK Discussion
Why is it important to consider social and ethical issues when developing source code? Surely once the code is compiled nobody will ever know how it was developed. Discuss.

ERRORS OCCURRING IN SOFTWARE SOLUTIONS

There are three different types of error that can occur. Syntax errors will be detected when the code is translated. Logic errors and runtime errors are realised as the code is executing. We now examine each of these three types of error:

Syntax Errors

Earlier in this chapter, we examined the translation process in detail. The translation process will find any syntax errors that exist in the code. Syntax errors are any errors that prevent the translator converting the high-level code into object code. Any error resulting from incorrect use of the high-level language is called a syntax error. Syntax errors result from both the lexical analysis stage and the syntactical analysis stage of translation.

Errors from the lexical analysis stage result from sequences of characters that cannot be represented as tokens. For example, the sequence of characters MoveOn! is not a legitimate identifier in most languages as the exclamation mark is neither a letter or a digit. It cannot be tokenised so a syntax error is generated.

Most syntax errors result from the process of syntactical analysis (hence the name syntax error). For example, the Pascal statement WHILE Count<20 would create a syntax error because the reserved word DO must follow the condition in a Pascal pre-test statement.

Most modern programming environments check the syntax of statements as they are entered. The programming environment contains a mini translator that translates each line of code as it is entered to specifically check for syntax errors.



Fig 5.25
Syntax errors are detected as the code is entered in many programming environments.

Once a syntax error has been detected and identified, it is usually a simple task to correct the error. Many interpreters and compilers inform the programmer as to how the error may be corrected.

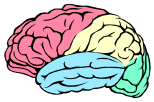
Logic Errors

Logic errors are the result of syntactically correct code that does not complete the required task. Logic errors can manifest themselves in various forms. Continuous loops will cause the computer to appear to freeze. One incorrect calculation in a subroutine can cause errors in routines above, which in turn cause further errors in other routines resulting in incorrect output. Logic errors can cause the program to halt, in which case a runtime error is generated.

Algorithms aim to explain the logic required in a software solution. Errors in algorithms will manifest themselves as logic errors in the final code. Thorough testing of algorithms is necessary to minimise the number of logic errors encountered at the implementation stage. Even if the algorithms are correct, the process of converting them into a particular programming language can introduce logic errors.

Logic errors are the most difficult type of error to correct. It is impossible for the development environment to detect logic errors let alone suggest methods of correction. The computer cannot possibly know the intention of the programmer; it merely follows the set of instructions it is given. If the instructions are correct in terms of syntax then the computer will happily carry them out.

The techniques and software tools discussed in the next section are aimed at assisting the programmer to detect and identify the source of logic errors. Once identified, the programmer themself must correct the incorrect logic.



Consider the following:

A function is required that adds up counting numbers. For example, the call SumInts(4) should return the value 10 which is 1+2+3+4. The function executes with no errors reported, however the result returned is not correct. Clearly, the function contains one or more logic errors.

```
Function SumInts(SumTo:Integer):Integer;
VAR
Counter,Sum:Integer;
BEGIN
   Counter:=0;
   Sum:=0;
   WHILE Counter >=SumTo DO
      BEGIN
         Counter:=Counter+1;
         Counter:=Sum+Counter;
      END;
   SumInts:=SumTo
END;
```

Fig 5.26
Pascal function to add up counting numbers. (Including logic errors).



GROUP TASK Activity
Identify and correct the logic errors in the Pascal SumInts function shown in *Fig 5.26*.

Runtime Errors

Runtime errors are errors detected by the computer whilst a program is executing. There are many possible reasons for the system to generate a runtime error. The error, or bug, can be linked to either hardware or software problems. As we are primarily interested in software, let us consider the possible source of software runtime errors:



Bug
An error or defect in software or hardware that causes a program to malfunction. Apparently a real moth trapped between two relays caused the first computer bug.

- BIOS (Basic Input Output System)

The BIOS provides the interface between the operating system and the hardware. BIOS settings are stored in CMOS (Complementary Metal Oxide Semiconductor). CMOS is a type of RAM that requires very little power to retain its contents. A battery in the system box supplies power to the CMOS when the computer is turned off. Incorrect CMOS settings will often result in runtime errors.

- Operating system

Runtime errors generated by the operating system can result from various sources. Often BIOS, hardware driver or application software errors will cause an operating system runtime error. The all too familiar MS-Windows family blue screen is the result of an operating system runtime error.



Crash
A serious computer failure. The computer itself stops working or an application aborts unexpectedly. Caused by hardware malfunction or a serious software bug.

- Hardware drivers

Hardware drivers are normally loaded by the operating system. These are programs that provide an interface between the operating system and any hardware devices

installed on the computer. For example; hard disks, floppy disks, CD-ROM drives, sound cards, monitors, printers, scanners, modems, etc. The hardware driver itself can generate runtime errors, in which case a total system crash does not occur. If the problem is more substantial an operating system runtime error will occur, in which case it is likely the computer will crash.

- Application software

Runtime errors generated by application software can have a number of causes. Arithmetic overflow, division by zero errors and accessing inappropriate memory locations are three common possible causes.

An arithmetic overflow error occurs when a value is assigned to a variable that is outside the range allowed for that data type. For example, in many programming languages a variable declared as an Integer has a range from -32768 to 32767. Trying to assign a value outside this range to an identifier declared as an integer variable will result in an overflow error.

Division by zero is undefined in mathematics hence a runtime error is generated if this situation occurs. Often unexpected inputs result in division by zero errors. For example, calculating the average of a list of numbers will result in a division by zero error if the list is empty.

Accessing inappropriate memory locations occurs when an attempt is made to assign a value to an identifier that either does not exist or is of a different data type. For example, if an array called ThisArray is declared with a subscript range of 0 to 10, an attempt to assign a value to ThisArray(11) will cause a runtime error. Usually the error message will be something like 'Subscript out of range'. If we have a variable of integer type called MyInteger and another called MyString declared to hold characters then assigning MyString to MyInteger will result in a runtime error such as 'Type mismatch'.

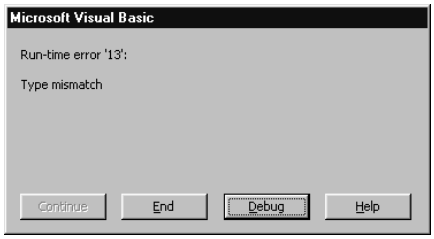


Fig 5.27
A runtime error message indicating a type mismatch has occurred.

Various other runtime errors are possible, many occurring as a result of a logic error in the code. For example, stack overflow errors often occur when a subroutine inadvertently makes a recursive call. A recursive call is when a subroutine calls itself. Each time a function is called an entry is placed onto the stack. Recursive calls can result in an attempt to place an infinite number of calls onto the stack resulting in a stack overflow runtime error.

Runtime errors resulting from the current project can often be anticipated. Most programming environments allow the program to intervene when a runtime error occurs. Exception handling is a common technique used by many languages. Subroutines should be written to anticipate and deal with runtime errors that may occur. Error subroutines should attempt to correct the problem; if this is not possible, they should attempt to save any data files before exiting.



GROUP TASK Investigation
Explore the help files in a programming environment with which you are familiar. Determine how the language deals with the detection and avoidance of runtime errors.

TECHNIQUES FOR DETECTING ERRORS IN CODE

Many techniques are available to the software developer to assist in the creation of error free source code. In this section, we examine a number of techniques commonly used during the implementation stage of the software development cycle. Each method aims to reduce the number of logic errors in the final code.

We will examine the following techniques:

- Stubs
- Drivers
- Flags
- Debugging output statements
- Peer checking
- Desk checking
- Structured walkthrough
- Use of expected output

Stubs

A stub is a small subroutine used in place of a yet to be coded subroutine. The use of stubs allows higher-level subroutines to be tested. Stubs do not perform any real processing; rather they aim to simulate the processing that will occur in the final subroutine they replace. Stubs are used to set the value of any variables affecting their calling routines and then end. Sometimes a stub may include an output statement to inform the programmer that the call to the stub has been successful.

The creation of stub subroutines is required when software projects are coded using a top-down design methodology. Because higher-level subroutines are created before lower-level subroutines, it is necessary to create 'dummy' subroutines. This enables testing of the higher-level subroutines whilst they are being coded.

Drivers

A driver, in general terms, provides an interface between two components. A driver controls the operation of some device. For example, a hardware driver is a program that provides the link between the operating system and a specific hardware device. Another example is a printer driver used to control the operation of a printer.



Fig 5.28
The driver of a car provides an interface between the environment and the car. The driver controls the operation of the car.

In terms of software development, a driver is a subroutine written to test the operation of one or more other subroutines. Commonly, drivers set the value of any required variables, call the subroutine to be tested and output the results of the subroutine call. In essence, the driver is an interface between the subroutine and the programmer. The driver controls the subroutine.

Drivers are required when software projects are coded using a bottom-up design methodology. Lower-level subroutines are developed before higher-level subroutines. Because of this, it is necessary to write a driver to test the operation of each subroutine as it is being coded.

Flags

A flag is used to indicate that a certain condition has been met. Usually a Flag is set to either true or false, that is, most flags are Boolean variables. We have examined the use of flags within algorithms in chapter 4 and then again in relation to the operation of the CPU earlier in this chapter. In each case, flags are used to signify the occurrence of a certain event.

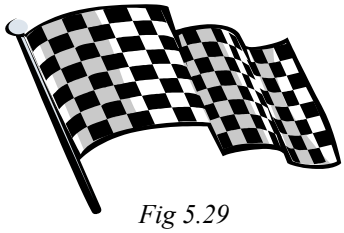


Fig 5.29
A black and white chequered flag is used to signify the end of a motor race. Either the race is over or it is not.

In terms of error detection, flags are used to check if certain sections of code have been executed or certain conditions have been met. For example, if a subroutine is called, a particular flag may be set to true. By examining the value of the flag, the programmer can determine the flow of control through the program. Setting the value of flag variables at different points in the code assists the programmer with the detection of the source of bugs.

Debugging output statements

Strategic placement of temporary output statements can help to isolate the source of errors. By placing output statements within called subroutines, the programmer can determine which subroutines have been called. This assists in the detection of the source of an error. Often a debugging output statement will be progressively moved as the debugging process continues. In this way, the flow of execution through the code can be precisely monitored. Eventually the source of the error is detected.

Often output statements will be used to display the contents of variables at crucial stages during execution. This allows the programmer to monitor changes in the contents of variables whilst the code executes. The precise place in the code where an erroneous calculation has occurred can then be determined.

Output statements are often used in conjunction with stubs, drivers and flags. In these cases, the output statement is used to indicate a particular event has occurred. A stub may contain an output statement to confirm it was actually called. Drivers usually contain statements to output the results of a call to a subroutine and the value of flags needs to be displayed during execution.



GROUP TASK Discussion
You are writing a function that sorts an array of data. Unfortunately all is not well, your subroutine is not working as expected. Stubs, drivers, flags and debugging output statements are all methods of error correction. Discuss how each of these techniques could be used to isolate the source of the error(s) in your sort function.

Peer Checking

Often errors will occur that seem to be impossible to correct for the original programmer. Colleagues on the development team are often able to see the problem from a fresh point of view. Peer checking involves other team members in the task of analysing each other's work to detect and correct errors.

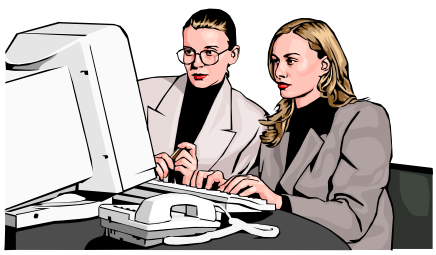


Fig 5.30
Informal peer checking is crucial to the successful development of software products.

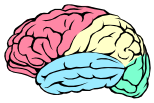
Many software development companies require that each subroutine is 'peer checked' as part of their quality assurance strategy. Errors that go unnoticed or uncorrected and are released to the public can cost software companies both in terms of direct financial costs as well as customer satisfaction. Formal peer checking involves programmers formally signing each other's work to verify they have thoroughly checked each coded subroutine.

Informal peer checking is used by virtually all programming teams. The flow of comments and ideas between team members is crucial to the successful development of any software product. This is particularly true in the case of detecting and correcting errors.

Desk checking

Desk checking is the process of working through an algorithm or piece of source code by hand. A table with a column for each variable is used. As the algorithm or code is worked through by hand, changes to variables are made by writing the new value under the appropriate identifier.

A desk check is particularly useful when the workings of a piece of source code are not fully understood. The process of working through the code statement by statement helps to make the logic clear. Often errors in the code will be detected as the desk check progresses. These errors can then be corrected and a new desk check commenced.



Consider the following:

A function is under development that will perform a linear search for a particular element amongst an array of elements. The function returns the value true if the item is in the list and false if it is not. The function has been written in Visual Basic but contains errors. *Fig 5.31* shows the function in its current state.

The programmer decides to perform a desk check of their code using the following test data array.

```
Animal(0) = "Frog"
Animal(1) = "Dog"
Animal(2) = "Cat"
Animal(3) = "Cow"
Animal(4) = "Ant"
Animal(5) = "Moose"
```

They decide to perform the desk check four times using the following four calls:

```
Linear(Animals(),"Frog")
Linear(Animals(),"Cat")
Linear(Animals(),"Moose")
Linear(Animals(),"Bat")
```

```
Public Function Linear(Search(), FindItem) As Boolean
Dim Lower, Upper, Count As Long
Dim Found As Boolean
Found = True
Lower = LBound(Search)
Upper = UBound(Search)
Count = Lower
While Count < Upper
   If Search(Count) = SearchItem Then
      Found = True
      Count = Upper
   End If
   Count = Count + 1
End While
Return Found
End Function
```

Fig 5.31
Linear search Visual Basic function with errors.

The desk check for the Linear(Animals(),"Frog") call is reproduced below:

Found	FindItem	Lower	Upper	Count	Search(Count)
True	Frog	0	5	0	Frog
True				5	
				6	

The code correctly finds Frog and returns the value true.

Let us now perform the desk check with the second test call Linear(Animals(),"Cat"):

Found	FindItem	Lower	Upper	Count	Search(Count)
True	Cat	0	5	0	Frog
				1	Dog
				2	Cat
True				5	
				6	

Again, the call correctly returns the value true.



GROUP TASK Activity

Complete the desk check with the remaining two calls:
Linear(Animals(),"Moose") and Linear(Animals(),"Bat")

There are actually 2 errors in this function. What are these errors and how can they be corrected?

Structured walk through

Structured walk throughs, as the name suggests, are more formal than peer checks. The developer or team of developers present their work to a group of interested parties. This group may include representatives from management, marketing and potential users, with the aim being to present the software and formally work through its functionality. The developers walk the group step-by-step through each aspect of the program. As the walk through continues, comments are written down for future consideration. No attempt is made to correct or justify aspects of the product; the aim being to receive feedback on the product as it stands.

Structured walk throughs are normally undertaken as formal meetings. Each person in attendance at a structured walk through should be given all relevant documentation prior to the meeting. This allows them to have an overall view and feel for the product and its design. The walk through itself should be a demonstration of the product and its design. Comments can either be made verbally or may be restricted to written comments. In either case, a response to comments should not occur during the walk through, rather at a later time when they can be considered carefully.

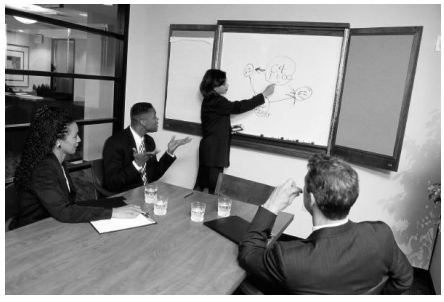


Fig 5.32
Structured walk throughs are normally undertaken as formal meetings.

Structured walk throughs can be used to evaluate the design at different levels. Their aim is to explain in a structured manner the operation of some part of the design and development process and to obtain feedback from interested parties. They may be

used to walk through the logic within an algorithm, the source code written for a specific module or they may be used to evaluate the final product once it has been implemented in code. In fact, most aspects of the software design and development process can be evaluated using structured walkthroughs.



GROUP TASK Discussion
Why do you think comments and suggestions should be collected but not responded to until after the walk through is complete? Surely it would be better to discuss any issues during the walk through. Discuss.

Use of expected output

When initial test data items are created the expected output from these inputs should be calculated by hand. Once the subroutine has been coded, the test data is entered and the output from the subroutine is compared to the expected outputs.

If the actual and expected output match then this provides a good indication that the subroutine is performing accurately. If they do not match then clearly a logic error has occurred. Further techniques must be employed to determine the source of the error.

If the test data has been designed to test every path through the source code and every boundary condition then a 100% match between actual and expected output means the source code is logically correct. It does not necessarily mean the source code is efficient, high quality or free of runtime errors.

For large programs, it is impractical to test every path through the entire program. A typical application can contain many thousands of lines of code. The number of test data sets easily runs into the billions. If subroutines have been designed using the 'one logical task per subroutine' rule, then it should be possible to perform the test on each subroutine.

Currently test driven development (TDD) methodologies are becoming popular. When TDD is used then thorough tests (together with test data) are written prior to coding. Passing these tests indicates the source code is achieving its purpose. As new subroutines are coded new tests are added, all these tests can be rerun each time new or modified code is added to the project. Often the source code for the test routines can be as large as the source code for the actual project.



GROUP TASK Activity
'If every path and every boundary condition has been tested and the output matches exactly the expected output then surely the code must be free of errors.' This statement is not true. Can you give examples of errors in code that will not be found by testing each path and boundary?



GROUP TASK Research
Many universities now teach computer science using a test driven development (TDD) methodology. Research using the internet or otherwise to find examples of teaching resources used by computer science departments in universities which demonstrate TDD.

SOFTWARE TOOLS FOR DETECTING ERRORS IN CODE

Most software development environments include tools to assist the programmer in the task of isolating errors in the source code. The majority of environments include automatic syntax checking as statements are entered. Different environments will provide different tools to assist in the identification and isolation of logic and runtime errors.

The following five tools are commonly available to assist in the detection of logic and runtime errors:

- Breakpoints
- Resetting variable contents
- Program traces
- Single line stepping
- Watch expressions

Breakpoints

Breakpoints are used to temporarily halt the execution of the code. Once execution has stopped it is possible to examine the current value of each variable. By adding breakpoints at strategic points within the code, it is possible to locate the source of the error.

Setting breakpoints is performed in one of two ways. Many environments provide a 'break' statement, which is entered in the code where a break is desired. Other environments provide a menu item that inserts a breakpoint at the current cursor position.

In Visual Basic, once a break point is reached execution halts (see *Fig 5.33*). By holding the mouse over identifiers in the source code, the current value held in the variable is displayed. An immediate window allows for code to be entered for instant execution.

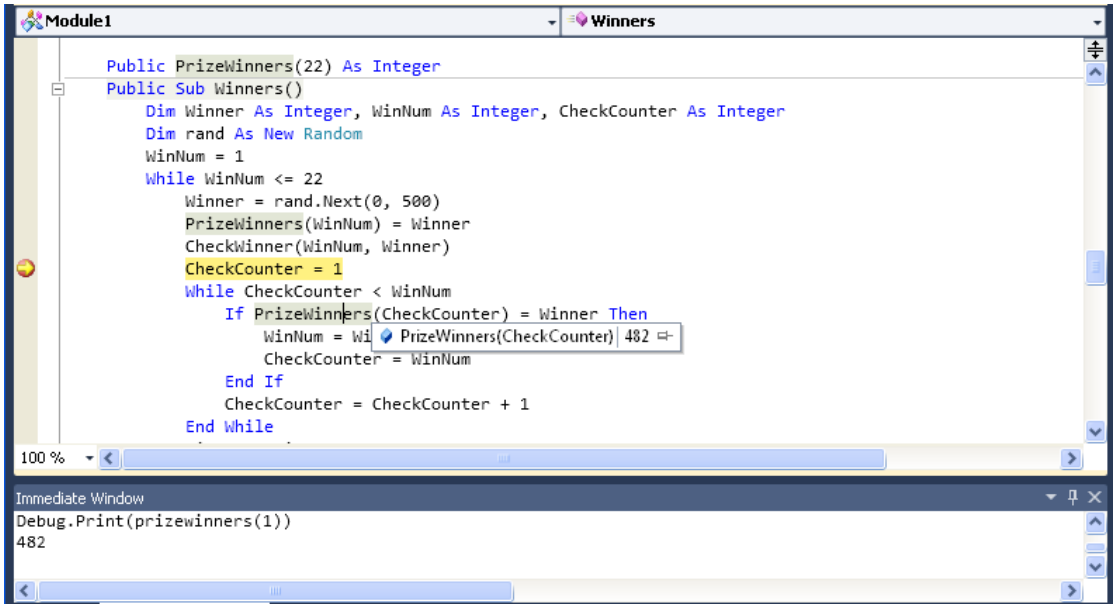


Fig 5.33
The Visual Basic (VB.NET) environment in break mode.

Resetting variable contents

Altering the value of variables during the execution of code can be useful in determining the nature of an error. The ability to alter a variable's contents can also allow the programmer to determine which values are causing errors without the need for the variable to actually attain that value as a consequence of execution.

Often the contents of a variable will be changed whilst execution has been halted by a breakpoint. For instance, in *Fig 5.33* we see that PrizeWinners(1) = 482. In the immediate window we could enter an assignment statement to alter the value of PrizeWinners(1) to a different value to test a particular scenario.

Inserting assignment statements within the source code itself can also be used to temporarily alter the contents of variables. For example, if the programmer wishes to ensure a particular piece of code is executed then inserting a temporary assignment statement can force execution to progress in a particular direction. Perhaps a particular calculation is causing problems, variables included in the calculation can be reset to differing values to allow the programmer to thoroughly analyse the calculation.

Program traces

Tracing, in terms of error detection, refers to tracking some aspect of the software. A program trace enables the order of execution of statements to be tracked or the changing contents of variables to be tracked. Both these types of trace involve analysing and following the flow of execution.

Various forms of trace are possible. Some programming environments can display each line of code as it is executed at slow speed. Others allow you to analyse the call stack when a breakpoint is encountered. The call stack is a list of subroutine names from the current call back to the main program. Many COTS products, particularly data oriented products, are able to produce a trace file. This file maintains a log of all the transactions that have taken place. This trace log can be analysed to identify the source of an error.

Many development environments provide a window containing all the variables local to the current subroutine together with their current value and data type. This list is updated each time the contents of a variable changes. This window is used in conjunction with breakpoints or single line stepping. Examining the changing contents of variables can assist in tracing the source of errors.



GROUP TASK Discussion

Division by zero and overflow runtime errors are highlighted by the system where they occur. Often the calculation that ultimately caused the runtime error is not at fault. Discuss methods of identifying the true source of these errors using breakpoints, resetting variable contents and program traces.

Single line stepping

Single line stepping is the process of halting execution after each statement is executed. Each statement is highlighted as it is executed. A simple keystroke allows execution to continue to the next statement. For example, in VB.NET the F8 key steps through the code line by line.

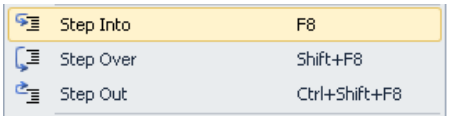


Fig 5.34
Stepping tools in VB.NET

Single line stepping can become somewhat monotonous when a large number of procedure calls or loops are involved. Most environments provide variations on the single line stepping idea. Commands to allow execution to step over procedure and function calls assist in isolating an error to a particular subroutine. Step out options mean that procedure and function calls are completed before execution once again halts. This style of stepping allows the programmer to concentrate on code within the higher-level subroutine.

Watch expressions

A watch expression is an expression whose value is updated in the watch window as execution continues. Usually watch expressions are variable names. In *Fig 5.35* the variable CheckCounter, Winner and PrizeWinners have been entered as watch expressions. Watch expressions can also be calculations such as Price + GST.

By combining single line stepping and watch expressions, an automated desk checking system can be created that can be used to watch the value of particular expressions. The watch window is similar to an individual row of a manual desk check.

Watch

Name	Value	Type
CheckCounter	4	Integer
Winner	54	Integer
PrizeWinners	{Length=23}	Integer()
(0)	0	Integer
(1)	234	Integer
(2)	70	Integer
(3)	216	Integer
(4)	490	Integer
(5)	28	Integer
(6)	166	Integer
(7)	232	Integer
(8)	54	Integer
(9)	0	Integer
(10)	0	Integer
(11)	0	Integer
(12)	0	Integer
(13)	0	Integer
(14)	0	Integer
(15)	0	Integer
(16)	0	Integer
(17)	0	Integer
(18)	0	Integer
(19)	0	Integer
(20)	0	Integer
(21)	0	Integer
(22)	0	Integer

Fig 5.35
Watch window in VB.NET displaying the current value of watched variables and expressions as the code runs.



GROUP TASK Discussion

A subroutine is under development that reads a sequential file of records. As each record is read various calculations are made. A logic error exists, as some of the calculations on particular records are incorrect. Explain how the use of single line stepping and watch expressions could be used to help isolate the problem.



GROUP TASK Activity

Use a programming environment with which you are familiar to perform the following: Write a short program to find the average of 10 random numbers between 1 and 100 inclusive. Create watch expressions for each of the variables in your code. Step through the code line by line to ensure it is operating correctly.



HSC style questions:

1. A new software application has been installed. A number of users report that the result of a calculation performed by the application is incorrect. What type of error has occurred?

(A) Logical
(B) Syntax
(C) Runtime
(D) User

Refer to the following algorithm when answering Question 2 and 3. Line numbers are included to assist when answering question 3.

```
10   BEGIN
20         numberOfStudents = 0
30         totalStudentMarks = 0
40         INPUT studentMark
50         WHILE studentMark >=  0
60               totalStudentMarks = totalStudentMarks + studentMark
70               numberOfStudents = numberOfStudents + 1
80               INPUT studentMark
90         ENDWHILE
100        averageMark = totalStudentMarks / numberOfStudents
110        PRINT averageMark
120  END
```

2. If the above algorithm is implemented and the first mark entered is negative then an error occurs. What type of error is this?

(A) Arithmetic overflow error
(B) Division by zero error
(C) Logic error
(D) Syntax error

3. Which of the following additional lines best corrects the "first mark entered is negative" error?

(A)
```
95   numberOfStudents = numberOfStudents + 1
```

(B)
```
95   IF numberOfStudents <> 0 THEN
115  ENDIF
```

(C)
```
95   IF studentMark < 0 THEN
115  ENDIF
```

(D)
```
93   IF numberOfStudents = 0 THEN
94         numberOfStudents = 1
95   ENDIF
```

Question 4.

As part of a software company's ongoing quality assurance procedures the following rules are followed by all programmers as they code subroutines.

- Global variables are not to be used.
- Subroutines must perform a single logical task.
- Completed subroutines must be peer checked prior to inclusion.

Explain why these rules will likely improve the quality of the company's products.

Suggested Solutions

1. (A) The code operates without displaying an error hence it cannot be a runtime or syntax error. The error is in the logic underlying the calculation.

2. (B) The body of the loop is never executed hence numberOfStudents is 0 when the division is attempted in line 100.

3. (B) Answer (B) effectively jumps the calculation and display of the average (lines 100 and 110) when there are no students. Although (D) also solves the problem it is not mathematically correct to display an average of zero when there are no student marks entered.

Question 4.

The widespread use of Global variables means that the use of the same variable names as a local variable will cause conflict and developers can never be sure where global values have been changed. If no global variables are allowed at all, then all data that needs to be shared between routines will need to be passed as parameters between the routines concerned. In this way, the different programmers involved in the development of the software have much more flexibility in their choice of variable names without fear of ambiguity or conflict arising when the various modules are pulled together into the final product.

When each subroutine performs one logical task only and is named appropriately to reflect that task, it is much easier to locate the source of logic errors during the testing phase. It is also easier for peer checking as the logic is much easier to follow by a programmer who is not the person who originally wrote it. In addition, future maintenance programmers can be more confident when they improve the processing of a routine as the job it performs is a single clear task.

The requirement to have each routine checked by another programmer prior to including it into the final version of the code helps ensure the code is error free, as well as ensuring that the source code adheres to the standards expected by the company. It also means that the final released version of the source code is more coherent, without each subroutine reflecting the individual preferences of the developer who wrote it. This makes it much easier to maintain such code, whether it is to customise it for a particular company or to remove errors found later.



GROUP TASK Discussion

Question 4 lists three rules. Suggest other possible rules or guidelines that would also assist a software company's ongoing quality assurance.

SET 5D

1. Many techniques are available to reduce the number of errors in the final solution. The most important technique is:
   (A) to use a clear modular structure.
   (B) use of debugging output statements.
   (C) to use drivers and stubs.
   (D) peer and desk checking.

2. Which of the following is true of logic errors?
   (A) They are discovered during translation.
   (B) They are usually discovered at runtime.
   (C) They always cause a halt in execution.
   (D) They result from misspelt reserved words.

3. The difference between peer checking and desk checking is:
   (A) Peer checking is the process of working through the code by hand. Desk checking is performed automatically by the computer.
   (B) Desk checking is the process of working through the code by hand. Peer checking involves analysing and checking code written by others.
   (C) Peer checking is used by large software developers to ensure code is error free. Desk checking is used to ensure algorithms are logically correct.
   (D) Desk checking and peer checking are different names for essentially the same process.

4. Constants used throughout a program should be assigned to an identifier. Why is this the case?
   (A) The value need only be changed in one place by future maintenance programmers.
   (B) The code executes faster.
   (C) Calculations will be easier to understand.
   (D) All of the above.

5. Which of the following is true of syntax errors?
   (A) They are always discovered during translation.
   (B) They are always discovered at runtime.
   (C) Can result in unexpected outputs.
   (D) They always result from misspelt reserved words.

6. Runtime errors are detected whilst the program is running. They can be the result of:
   (A) hardware faults.
   (B) faults in the operating system, BIOS or hardware drivers.
   (C) application software errors.
   (D) All of the above.

7. If a program is coded using a top-down design methodology, which of the following would be required to test each module?
   (A) flags.
   (B) stubs.
   (C) drivers.
   (D) break points.

8. Quality solutions:
   (A) perform the stated task without error.
   (B) will solve a single logical task.
   (C) are superior in efficiency, maintainability and design.
   (D) provide superior documentation to other solutions.

9. A temporary routine that calls other subroutines to assist in their testing is called a:
   (A) stub.
   (B) driver.
   (C) flag.
   (D) main program.

10. Temporarily halting the execution of code so that the contents of variables can be examined and altered is accomplished using:
    (A) watch expressions.
    (B) single line stepping.
    (C) program traces.
    (D) break points.

11. 'Coding of a software solution can be done from the top down or from the bottom up.' What does this mean? What techniques are required when coding using each of these methodologies?

The Pascal code fragment that follows, attempts to find all the factors of a number. Factors are numbers that divide evenly into a number. For example, the factors of 8 are 1, 2, 4 and 8. The TRUNC function returns the integer part of a number. This code fragment contains a number of errors.

```
Readln (Number);
Count := 0;
WHILE Count < Number DO
  BEGIN
    IF Number/Count = TRUNC(Number/Count)
      THEN Writeln (Number/Count);
    Count := Count + 1
  END;
```

12. Perform a desk check of the above code using 12 as the input. What errors are encountered? Suggest ways of correcting these errors. Perform the desk check again with your corrected code.
13. The above problem can be solved in a much more elegant and efficient manner. Make changes to the code to make it more elegant and efficient.

The modules for a large program have been developed separately. Apparently, all subroutines in each module has been thoroughly tested on its own. These modules are now being combined to form the total solution. A number of errors are occurring when the code is executed.

14. List and describe the types of errors that could be occurring. Give an example of each possibility.
15. For each of the error types identified in question 14, describe a possible technique for isolating the error.

DOCUMENTATION OF A SOFTWARE SOLUTION

Throughout the development of a software solution, various forms of documentation are created. We have already discussed the need for continually updating documentation throughout the development process. In this section, we examine the total documentation required before software solutions are released to clients.

Documentation can be split into two broad categories: user documentation and technical documentation. These categories describe the intended audience of the documentation. The nature of the software product and its intended audience will determine the language and complexity of each documentation item.

USER DOCUMENTATION

Users are the final operators of the product. Documentation targeted at users needs to be directed at their level of knowledge and expertise. A game designed for use by children will require very different user documentation to a product designed to automate the operations of a large corporation. The end users of the system must be identified before the production of user documentation commences.

There are many types of user documentation possible. In this section, we examine four specific types of documentation that are provided with the majority of software products. Each of these documentation types is commonly provided in either printed or electronic formats.

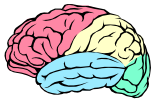
Installation Guide

Installation guides should provide the user with sufficient information to successfully install the software product on their machine(s). Hardware and software requirements will be included together with step-by-step instructions to guide the user through the set-up process. Often notes are included to help resolve common installation problems.

Hardware requirements should be clearly stated. Often a minimum set of hardware requirements will be given together with a recommended set of requirements. Many products require other software to be installed on the system prior to installation, operating systems being the most obvious example.

The main section of the installation guide describes the steps that need to be undertaken to install the product. For small applications, this can be as simple as placing a CD-ROM in a drive and waiting for the installation program to auto-run. Large systems often require significant direction from the user, particularly in regard to configuring the system for the particular installation site. For large systems an installer from the software company, or one of its distributors, often performs or directs the installation process.

Whilst is should not be expected, it is not uncommon to encounter errors during the installation process. The large variety of hardware and software combinations available cannot all be tested. As a consequence, installation errors can occur. The installation guide should attempt to address the most common of these. A good installation guide will also provide support contact details in the event that the user cannot resolve an installation problem.



Consider the following:

A chain of restaurants is installing a computer system to improve the productivity of their processes. The software has been written by HospSys, a software development company specialising in software solutions for the hospitality industry. The restaurants will be connected to their head office via dedicated modem connections. The software has been written and an installation guide is currently being prepared. An installer employed by HospSys will complete the actual installation of the software. The installer will also provide the initial training to users.

There are two audiences involved in this installation: the restaurant personnel and the installers from the software company. Specific sections of the installation guide are directed at each of these two groups. The installer will have a copy of the entire installation guide whereas the restaurant personnel require items pertinent to their particular installation tasks.

Let us examine items that are included in the restaurant personnel section of the installation guide:

Hardware and software requirements

- File server running Windows Server detailing minimum disk and RAM requirements.
- Workstations for order entry
- EFTPOS terminal with dedicated phone line
- Manager's workstation and printer
- POS (Point of Sale) terminal with receipt printer
- Network hub and associated cabling
- Modem with dedicated phone line

Installation steps

1. Order/Receive hardware and software from external vendor.
2. Schedule cable installation for network.
3. Schedule hardware/network installation.
4. Organise and install two phone lines. One near file server and one beside POS terminal.
5. Liaise with bank and have EFTPOS terminal installed.
6. Inform HospSys and schedule an installation date. Complete application software installation. Schedule employees of restaurant for initial training. Training completed by HospSys installer. System considered Live.

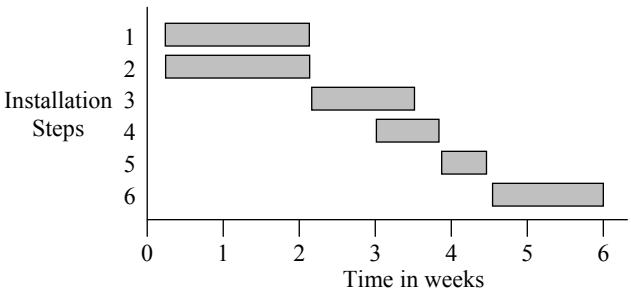


Fig 5.36
Gantt chart for the installation of the HospSys product.



GROUP TASK Discussion

The HospSys system is a client-server application. Each evening data from each restaurant in the chain is uploaded to head office. As part of the installation process the installer must ensure all aspects of the system are operational. Create a Gantt chart that could be included in the installer's section of the installation guide.

User Manual

Quality user manuals aim to provide concise and accurate information in regard to the operation and purpose of software. Topics in user manuals describe what tasks the software can complete, why these tasks are required and how these tasks are completed using the software. The information needs to be presented using a format that is appealing, accessible and consistent. Often much of the information traditionally contained in a user manual will be included in help files that are accessible directly from within the application.

User manuals can be organised in a number of ways depending on the nature of the software product. Topics can be ordered in terms of difficulty; in this way, users working through the manual progressively increase their skills. Topics can be presented in order of usage. Commonly used task are described first, with less common tasks described in the latter part of the manual. Software products containing a large number of tasks can be arranged in terms of function. For example, a user manual for an accounting package may contain topics on general ledger, accounts receivable and inventory and purchasing. Each topic describes the functionality of that particular aspect of the package.

Each topic within the user manual should provide two explanations of each of the software's functions; a conceptual explanation and a procedural explanation. Conceptual explanations explain what the function is and why the user may need to use that function. For example, a topic describing a save function explains what and why saving is necessary, for instance; 'Saving stores a copy of the file in a permanent location. Saving ensures the file is not lost if a power failure or runtime error occurs.' Procedural explanations describe how a process is performed using the software. For example, 'To save a file open the file menu and click on save.' Screen dumps are often used to assist with procedural explanations (see *Fig 5.37*).

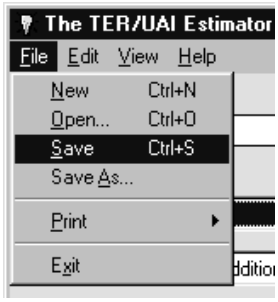


Fig 5.37
A screen dump displaying the Save command.

Reference Manual

Reference manuals are designed to be an efficient source of information. In terms of software, a reference manual should succinctly describe each command within the application. Reference manuals are not designed to be read from start to finish, rather they will be read in random order as needs arise.

Each command within the application should be listed in alphabetical order. This assists the user in locating required information without the need to refer to the contents or index pages. Normally each command is listed together with a brief description and a succinct example to illustrate its usage.

Often a quick reference card is included in the user's documentation package. This card lists all the commonly used commands together with shortcut keys and a brief explanation.

The ATAR Estimator
Quick Reference Guide

Menu Command	Shortcut Keys	Function
File – New	Ctrl-N	Create a new file
File - Open	Ctrl-O	Open an existing file
File – Save	Ctrl-S	Save the current file
File – Print – Individual	Ctrl-P	Print the current students results
Edit – Delete	Shift-Delete	Delete the current student record
Help – Using	F1	Open help
View – Individual	F2	Switch to individual student view
View – All Students	F3	Switch to all student view
View – By Course	F4	View students in a particular course

Fig 5.38
A quick reference card, showing commonly used commands and shortcut keys.

Tutorials

Tutorials provide instruction in the use of software products using example scenarios. The user is lead step-by-step through the processes included in the product. The user performs the tasks under the direction of the tutorial. Tutorials are designed so users can experience real world use of the application before using their own data.

Often sample data files are provided with applications for use during tutorials. These sample files are used to familiarise users with the functions available in the software product without the risk of damaging the real data. The use of sample files means that users do not need to input vast amounts of data to experience real life scenarios.

Tutorials should be split into individual lessons. Each lesson concentrates on developing a particular set of skills. Users can work through the tutorial at their own pace and as time constraints allow. Particular lessons can be undertaken, as the user requires the skills.

Paper-based tutorials, although still common, are being replaced by electronic versions. Usually a small window on the screen is used to direct the user through the tutorial.



GROUP TASK Investigation

User manuals, reference manuals and tutorials are the primary methods of providing instruction and information to users. Examine the written documentation included in various commercial software packages. Evaluate each item in terms of its intended function. How could these items be improved?

Online Help

Each of the above types of user documentation can be in either printed or electronic online form. It is now common for most user documentation to be provided online rather than as printed manuals. Online documentation can be provided as Adobe PDF files which are often similar in structure to more traditional printed manuals; however dedicated hypertext documents are now more common. Hypertext help documents allow users to efficiently search for specific items or in many cases they allow context sensitive help to be provided from within the application. When the user selects help within the application they are directed to the most relevant help topic automatically.

Online help can be truly online, meaning it is stored on a remote web server and hence requires an internet connection or it can be stored on locally on the user's machine. Currently many software applications only provide truly online internet based help. This has the advantage that the software company can update the help files as required and all users will immediately have access to the new content. In addition, interactive help such as user support forums and the ability to ask questions or suggest new functionality are valuable additions to an applications support options.



GROUP TASK Research

Consider a popular software application installed on your computer. Research and briefly describe all the various types of online help available to users of this software application.

TECHNICAL DOCUMENTATION

Technical documentation is designed for an audience who are proficient and knowledgeable in regard to the subject matter. In terms of software products, technical documentation primarily describes the structure and engineering behind a product.

Whilst software is being developed, thorough documentation should be maintained. These documents form the basis of the final technical documentation. Personnel involved in maintaining, upgrading and supporting software solutions make use of technical documentation.

Log Books (or process diaries)

A log book records the systematic series of actions that have occurred during the development of a project. Log books or process diaries are often utilised during the design and development of products in many industries. Maintaining a log book is a method of chronologically recording the processes undertaken to develop a final product. Individual members of the development team can maintain their own log books or a central log book can be maintained for each product under development.

The purpose of maintaining log books is twofold. Firstly, the information can be used to determine the relative cost of each stage of development. More importantly, log books are used as reflective devices. By examining the log books mistakes, delays or inefficiencies during development can be analysed. Future projects can then be more effectively managed to ensure such issues do not continue to arise.



Consider the following:

The following log book description is from the *Software Design and Development Stage 6 – Software and Course Specifications* document available on the NSW BOS website. This description refers to a log book you might use whilst working on a project as part of the SDD HSC course.

Log books

Log books are used to document the progress of a project. Entries should include:

- date
- description of the progress (or lack thereof) made since the last entry
- tasks achieved
- descriptions of stumbling blocks or issues encountered and how they were managed
- details of possible approaches for upcoming tasks
- reflective comments
- reference to resources used

Log books may be produced using spreadsheets, blogs, handwritten entries or electronic journal entries. Email messages to a fellow developer may be useful, as they contain time and date stamps. A sample log book entry may look like:

April 22nd 2010 – problems with images I am so pleased the coding for my mainline is finally done! This morning I spent some time importing the pictures for each screenshot. I had some trouble with it. The one problem I still have to fix is the transparency so I will try to import each in GIF format and make the background transparent. Hopefully it will work.

Note that the comment is reflective, describes what has been achieved, identifies a particular stumbling block and possible approaches to dealing with it.

Fig 5.39
Software Design and Development Stage 6 – Software and Course Specifications p22



GROUP TASK Discussion

Discuss how the regular use of a log book would assist in the development of software projects within the SDD HSC course.

Source Code Documentation

The source code itself is probably the most important form of technical documentation. It is virtually impossible to isolate logic errors in an application without access to the original source code. For source code to be a valuable technical resource for future maintainers, it must be intelligently documented.

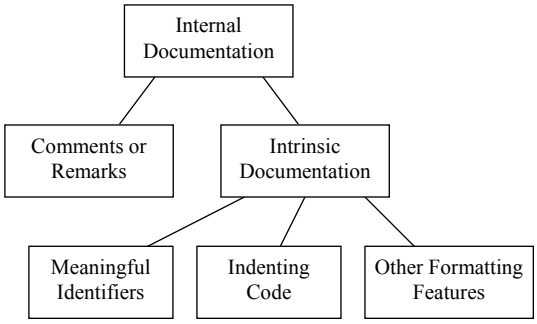


Fig 5.40
Source code documentation can take many forms.

Documentation within the source code is often called internal documentation. Internal documentation can take two forms, comments and intrinsic documentation. Let us examine each of these in turn.

- Comments

Comments provide information for future programmers. The translator ignores comments, or remark statements, within the code. Each procedure, function and logical process within the code should be preceded by a comment.

Comments should explain what a section of code does rather than how it does it. For example 'Find the largest value in the array' is a better comment than 'Make the first array value the largest then loop around for each remaining array item checking it against the current largest. If a value is larger it becomes the current largest value.' The second version of this comment is really echoing the code. Comments should summarise the code, enabling future programmers to grasp the intention at a glance. *Fig 5.41* is a Visual Basic 6.0 procedure. Reading the comments in this procedure explains clearly what the procedure accomplishes. The code itself provides information about how the processing is accomplished.

```
Private Sub RedGreenBlue(ColValue, Red, Green, Blue)
'ColValue is the input and is the 24 bit colour value.
'This procedure extracts the 0-255 value for each of
'the Red Green and Blue components and returns them
'in the parameters of the same name.
Dim HexCol
'First we convert the colour value to hexadecimal.
'The hex function returns a string. e.g. C05F3A.
HexCol = Hex(ColValue)
'Extract each colour value from the hex string.
Blue = Val("&H" & (Left(HexCol, 2)))
Green = Val("&H" & (Mid(HexCol, 3, 2)))
Red = Val("&H" & (Mid(HexCol, 5, 6)))
End Sub
```

Fig 5.41
Procedure containing comments describing the procedure and its logical tasks.

Comments that explain the logic of code should be used sparingly. Only for particularly difficult logic is a full explanation of the code's logic necessary. The code itself describes the logic. After all, our audience are programmers whom we presume can read code. Comments that explain the logic in detail can easily become incorrect as the code is modified.



GROUP TASK Investigation

In Basic a single quote is used to indicate a comment. In C++ two forward slashes are used. Examine a number of programming languages to determine how comments are indicated in the code.

- Intrinsic documentation

Intrinsic means belonging by its very nature. Intrinsic documentation is therefore documentation that is part of the code by its very nature. In other words, the code is self-documenting.

There are two main types of intrinsic documentation: the use of meaningful identifiers and indenting of code. Self-documenting code is easier for humans to read. The translator does not care what names are given to identifiers, just as long as they are syntactically legal. Translators ignore indenting.

```
PROCEDURE A; VAR B : Integer;
BEGIN B:=1; WHILE B<=10 DO BEGIN
WriteIn (Count); B:= B+1 END END;

PROCEDURE OneToTen;
VAR
   Count : Integer;
BEGIN
   Count:=1;
   WHILE Count<=10 DO
      BEGIN
         WriteIn (Count);
         Count:= Count+1
      END
END;
```

Fig 5.42
Intrinsic documentation makes source code understandable for humans.

An identifier is the name given to a variable, procedure, function or instance of an object. The design specification for the project will normally describe the naming conventions to be used within the project. In this text, we have generally used mixed case for identifiers. For example, in *Fig 5.42* the procedure is called OneToTen or could have been called oneToTen. Some developers use the underscore character to separate words, for example One_To_Ten. It is common practice to use upper case for all constant identifiers.

Identifiers should describe the purpose of the element it represents. If a variable is used to store client's names and addresses then appropriate identifiers could be: ClientDetails, ClientInfo or perhaps just Clients. Procedure and function names should describe the processing that occurs within the subroutine. For example, ExtractDate or SortArray.

Many programming languages have their own naming conventions. In Visual Basic, control names should begin with a mnemonic that identifies the particular type of control. Text box identifiers commence with txt, for example txtDate. Labels commence with lbl, for example lblResponse. List boxes with lst, combination boxes with cbo and so on. Adhering to these conventions enhances the readability of the source code.

Indenting is the process of setting lines of code back from the left margin. Indenting sections of code within control structures improves the readability of code. Many programming environments include automatic indentation settings. Consistent use of indenting allows the structure of the code to be seen at a glance. The beginning and end of each control structure stands out, this assists in visually describing the logic. The rules of pseudocode insist on the use of indentation, similar rules should be implemented within source code.

Many other formatting features are commonly used to improve the intrinsic readability of code. Leaving blank lines between comments and the code they describe. Colour can be used to visually differentiate between comments, reserved words, operators and operands. Ensuring each line of code can be viewed without the need for scrolling. Most programming languages provide a continuation character that is used to indicate that the statement continues on the next line. Any factors that increase the readability of the source code, and are part of the code, are classified as intrinsic documentation.



GROUP TASK Research

Search the Internet for code samples in a variety of different high-level languages. Comment on the intrinsic documentation used. What identifier naming conventions have been used?

Other forms of technical documentation

Throughout the software development cycle, various forms of documentation are created. These models, diagrams and documents should be revised and edited to reflect any changes made as the software development cycle has progressed. For example, the design specifications that were originally created in Stage 1 will no doubt have undergone modification and refinement as the project's development has unfolded. Any modifications that have been made should be documented and retained with the original specifications. CASE tools used during the software development process, assist in ensuring technical documentation is updated to reflect the final nature of the product.

System models, data dictionaries and algorithms must be particularly accurate. Future maintenance developers will require these documents.

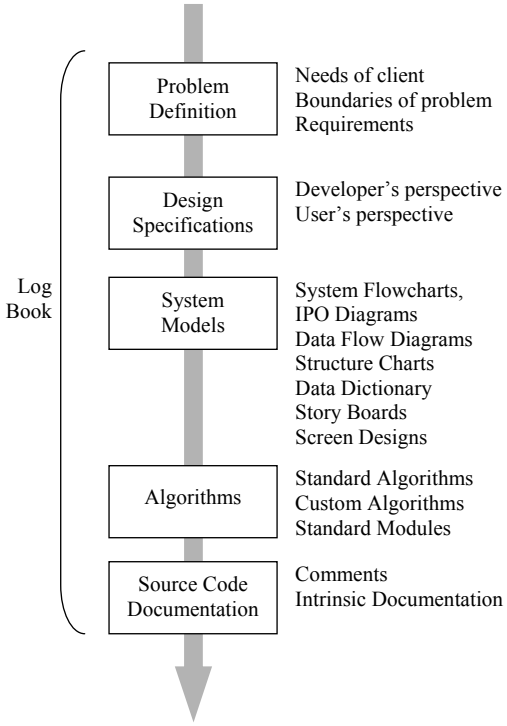


Fig 5.43
Technical documentation is compiled throughout the software development cycle.

The technical documentation we have examined in the SDD HSC course and that is created during the development of many software products is described in *Fig 5.43*.



GROUP TASK Activity

Make up a table listing all the possible technical documentation types studied so far in the text. Briefly describe each using diagrams as examples where appropriate.



HSC style questions:

1. Which of the following forms of documentation would be most useful to a maintenance programmer?
(A) Online help
(B) Gantt chart
(C) Story board
(D) Structure chart

2. Which of the following are all examples of user documentation?
(A) Installation guide, reference manual, system models and design specifications.
(B) Reference manual, tutorial, online help and intrinsic documentation.
(C) User manuals, tutorials, online help and installation guides.
(D) Online help, tutorials, system models and user manuals.

3. Which of the following best describes the purpose of internal documentation?
(A) To improve readability of source code.
(B) To improve the efficiency of code execution.
(C) To provide assistance to users of the software.
(D) To protect the intellectual property rights of the developer.

4. A software developer uses software that simulates many users entering data into their application simultaneously. This software is an example of which of the following?
(A) COTS package
(B) CASE tool
(C) Reverse engineering
(D) Test data

Question 5.

You have been contracted to make modifications to an existing software application that you did not write. Outline TWO examples of documentation that would be useful when making the modifications. Justify your choices.

Suggested Solutions

1. (D) **2.** (C) **3.** (A) **4.** (B)

Question 5.

Internal documentation incorporated into the source code which includes comments and also intrinsic documentation such as meaningful variable and procedure names. Internal documentation will make it easier for the programmer to understand exactly what the source code is doing, which makes it easier to determine which parts of the code need modifying to achieve the specified changes.

Systems documentation such as systems flowcharts, structure charts and DFDs provide an overview of the system and how the various parts interrelate. This provides the programmer with the information to determine the role played by individual components of the system, so they can better determine which parts need to be modified, and the effects of those modifications on other parts of the system.

SET 5E

1. Which type of documentation includes hardware and software requirements together with step-by-step set-up instructions?
   (A) User manual.
   (B) Reference manual.
   (C) Installation guide.
   (D) Tutorial.
2. Internal documentation is:
   (A) included in the source code.
   (B) comments used to explain the source code.
   (C) self documenting code.
   (D) All of the above.
3. The complexity of the language used in documentation should be determined by:
   (A) identifying the intended audience.
   (B) the nature of the product.
   (C) the complexity of the code.
   (D) examining similar products.
4. Examples of user documentation include:
   (A) installation guides, process diaries, reference manuals and user manuals.
   (B) installation guides, internal documentation, tutorials and user manuals.
   (C) user manuals, reference manuals, tutorials and system models.
   (D) user manuals, reference manuals, tutorials and installation guides.
5. Intrinsic documentation includes all of the following:
   (A) Meaningful identifier names, indenting within control structures, use of colour and spacing.
   (B) Comments, meaningful identifier names and indenting within control structures.
   (C) Internal documentation, algorithms and system models.
   (D) Objectives, data dictionaries, system models and feasibility studies.
6. A document listing all the commands available in an application is known as a(n):
   (A) User manual.
   (B) Reference manual.
   (C) Installation guide.
   (D) Tutorial.
7. A chronological record of the tasks undertaken during the development of a software solution could be best described as:
   (A) internal documentation.
   (B) a log book.
   (C) intrinsic documentation.
   (D) a system model.
8. A form of documentation that uses sample data to instruct the user in the use of a software product.
   (A) User manual.
   (B) Reference manual.
   (C) Installation guide.
   (D) Tutorial.
9. The sentence "The aim of this game is to capture all the opponents Greebs" is an example of a:
   (A) procedural explanation.
   (B) subjective explanation.
   (C) technical explanation.
   (D) conceptual explanation.
10. Comments within the source code are used to explain the code. Which of the following is true in regard to these comments?
    (A) They should explain the logic of the code that follows.
    (B) Each line of code should be preceded by a comment.
    (C) They should explain what a section of code does rather than how it does it.
    (D) Each comment should precisely reflect the processing of the code fragment that follows.
11. Make a list of all the user documentation types. Describe the main features of each.
12. Make a list of all the different forms of technical documentation. Describe the main features of each.
13. 'Documentation should be developed once the product is ready for distribution.' Do you agree with this statement? Explain your answer.
14. Electronic forms of user documentation are becoming commonplace. Explain the advantages of electronic forms of user documentation over traditional printed forms. Consider the point of view of the software developer and the user as part of your response.
15. Create a subroutine in a high-level language with which you are familiar, that splits a sentence into separate words. The words are to be stored in an array. Include suitable internal documentation as part of your solution.

HARDWARE ENVIRONMENT TO ENABLE IMPLEMENTATION OF THE SOFTWARE SOLUTION

All software solutions are designed with particular hardware requirements in mind. The installation guide will include specifications in regard to minimum requirements, possible additional hardware and appropriate drivers. Large custom systems are usually designed for a specific hardware configuration. Solutions designed for a broad market need to operate on a variety of hardware configurations. The hardware requirements need to be specified and tested before software solutions are released.

Minimum Hardware Configuration

The minimum hardware requirements will vary for different software solutions. The checklist, shown in *Fig 5.44,* lists hardware items that need to be considered by the software developer for all solutions. The minimum configuration should allow the software to operate successfully with acceptable performance and response times. *Chapter 6, Testing and evaluating software solutions*, examines how the hardware and software is tested to ensure performance meets expectations.

- Processor type and speed
- RAM size and type
- Secondary storage size and access speed
- Input devices
- Output devices
- Network hardware

Fig 5.44
Checklist of hardware items required for all software solutions.

Possible Additional Hardware

Additional hardware items are those items supported by the software product but not essential for its operation. Extra RAM may allow the product to operate with larger files. Sound cards may allow for optional audio feedback for the sight impaired. The addition of a network or internet connection may allow for foreign exchange rates to be updated electronically in financial packages.

Appropriate Drivers or Extensions

Hardware devices require software drivers to allow them to communicate with the system. These drivers are also known as interfaces, as they convert signals from one device into those that can be understood by another. Before a hardware device can be used by a software solution, its driver must be correctly installed and configured. Most peripheral devices come packaged with drivers for most popular operating systems or are able to use drivers included with the operating system. The driver is installed as part of the installation of the hardware device.

Custom built hardware may require a purpose built driver to be created. This driver would need to be installed as part of the installation of the software solution. In this case, the driver is an extension to the software package. For example, a product designed to monitor the climate within a green house includes custom drivers for the temperature and humidity sensors. These sensors are part of the minimum hardware requirements for this product. The drivers or extensions are installed as the main application is installed.



GROUP TASK Investigation

Examine the hardware specifications for a number of commercial software applications. Classify each as either a minimum requirement or an additional hardware item.

EMERGING TECHNOLOGIES

New advances in both hardware and software continue to have a profound effect on our daily lives. The implications of these technologies will have far reaching effects for software developers. Software developers must remain up to date with current and emerging technologies if they are to remain competitive. The Information Technology industry is renowned for its speed of evolution.

Let us consider some emerging hardware and software technologies that have the potential to alter both the human environment and the software development process.



Quantum Computers

Quantum theory postulates that matter can be in more than one state at the same time. For example, a mouse held in a sealed box could be either dead or alive. We don't know the state of the mouse until the box is opened. Quantum theory suggests that whilst in the box, the mouse is effectively both dead and alive. Strange as this may seem, science has found that photons, a component of light, actually behave in this manner. Photons behave as waves and at the same time behave as particles. Quantum physics defies the laws of classical physics, however it does explain the consequences of nuclear reactions and has helped unravel the mysteries of DNA.

The theory of quantum computing relies on spinning particles. Many particles at the atomic level are known to spin. The direction of the spin can be used to represent binary digits (bits). In quantum computing terms, bits are known as qubits (pronounced cubits). We are not only able to set the direction of the spin but we are also able to read the direction of the spin for an individual qubit. Using the 'mouse in the box' analogy, if we place particles in a box out of eyesight and apply energy, each particle will be in two states at the same time. If we have 8 particles in the box, we are representing 2^8 or 256 different states at the same time. Applying an algorithm to the particles allows calculations to therefore take place on all 256 states concurrently. If we expand our theory to many thousands or millions of particles, the potential processing ability becomes mind-boggling.

Although quantum computers are still very much a theoretical possibility, there are many companies investing heavily in their research and development. Why the investment? One very practical use revolves around security of data. Current methods of encrypting data rely on the fact that breaking the encryption is not possible with today's processing capabilities. It is said that if all the computers in the world worked continuously on breaking just one modern military encryption code it would take about the same amount of time as the universe has been in existence. Quantum computers could potentially break encryption codes in an instant.



GROUP TASK Discussion

Quantum computers, if they become a reality, will revolutionise the computing industry. How do you think the availability of virtually unlimited processing power will affect the software development industry?



Human-computer interaction

Research continues to improve the interaction between humans and computers, the aim being to simulate human communication using computers. Humans communicate using eye movements, facial expressions, touch and voice. These inputs combine to create each human message. The challenge for researchers is to create systems that are able to mimic this process.

Projects specialising in artificial vision are currently able to follow the eye movements of a user. Eye movements are analysed and used to move a cursor on the computer screen or alter the view appropriately as the user moves through a virtual world.

Users can direct computers and game consoles using body movements. Video cameras and other sensors are used to detect even small hand and finger movements. The software responds intuitively. For instance current dance games are able to detect timing and small body movements and shooting games can accurately determine the precise direction a person is pointing.

Output using gloves and full body suits incorporating pneumatic sensors is becoming a reality using lighter and more responsive systems. The computer opens and closes a series of valves within the glove or suit such that the user can feel objects as they are manipulated in the virtual world.

Voice recognition is continually being refined. For instance early mobile phone voice recognition was rudimentary and often failed to detect single words correctly. Today complete sentences are understood with amazing accuracy. Devices are under development that detect lip movements, causing directional microphones to home in on the vocal source. These devices are able to operate in high noise environments or where there are groups of users.



Fig 5.45
C-Pen text scanning pen.

The following devices all introduced new methods for human-computer interaction:

- Games console devices such as the Wii remote, PS3 move, Xbox Kinect which include various motion, video, sound and other sensors within an integrated system.
- Handheld communication devices such as the iPhone and other smart phones.
- Biometric devices such as finger print, iris, face recognition and other biometric scanning devices.
- Touch sensing devices including touch screens, multi-touch surfaces, movement sensors and touch pads.
- Scanning devices such as scanning pens for text, phone cameras used to scan QR (quick response) codes, smart card readers within EFTPOS terminals or RFID scanners used for stock control in warehouses, retail stores and libraries.
- Mind control devices such as Emotiv's Epoc Headset which is able to detect facial expressions, emotional state and cognitive processes using 14 sensors.



GROUP TASK Discussion

Examine the games consoles, smart phones and other devices owned by members of the class to identify all the input and output devices present.



GROUP TASK Activity

Identify all the various scanners you encounter during a typical week. Briefly explain how each scanner improves the ability of software to collect data compared to keyboard data entry.



GROUP TASK Discussion

Emerging methods of input and output will no doubt continue to have a profound effect on future software products. Discuss the effects you perceive these changes may have for the development of future software solutions.



Internet access initiatives

New devices and software applications are regularly released which allow more flexible access to the internet and which utilise the internet in many new and innovative ways.

Examples of newer internet related hardware and software initiatives at the time of printing include:

- Televisions, mobile phones, cameras, cars, white goods and other devices which increasingly include built in World Wide Web connectivity.
- Social networking software which operates on a wide range of devices and is integrated within a wide range of other software applications.
- Location based applications that utilise GPS coordinates and internet connectivity to map locations in real time.
- Wireless connectivity at ever increasing speeds which allows people to remain connected regardless of their location or activity.
- The National Broadband Network (NBN) which is planned to provide optical fibre to the premises of most homes and businesses throughout Australia.



GROUP TASK Discussion

Investigate new emerging internet technologies. How do these technologies affect human communication? What consequences are there for software developers working on internet solutions?



GROUP TASK Research

Use the internet to research new and innovative computer based devices. Write a paragraph on at least two such devices. Compile a class summary of all the devices found.

CHAPTER 5 REVIEW

1. Consistent user interfaces allow users to:
   (A) transfer skills to new products.
   (B) concentrate on the tasks at hand.
   (C) predict the results of their actions.
   (D) All of the above.
2. EBNF and railroad diagrams:
   (A) are used to describe algorithms.
   (B) describe the syntax of programming languages.
   (C) are important forms of user documentation.
   (D) can be altered by the programmer to suit the particular project.
3. Parsing is a process used during syntactical analysis. Parsing ensures:
   (A) each statement contains elements that are legal parts of the language.
   (B) source code is free of logic errors.
   (C) that source code cannot be changed by the end users.
   (D) each statement obeys the rules of the high-level language.
4. In regard to the role of the CPU in the execution of software, which of the following is true?
   (A) Assembler code is the native language of the CPU. Each assembler instruction corresponds to a precise hard-wired process.
   (B) Each high-level language calculation corresponds to one implementation of the fetch-execute cycle.
   (C) Microcode instructions are used to activate one implementation of the fetch-execute cycle.
   (D) The control unit translates high-level instructions into microcode.
5. The source code must be distributed with a particular product. What is the most likely reason for this?
   (A) The program must be translated using a compiler.
   (B) The software needs to be modified by the user.
   (C) The program is written for translation using an interpreter.
   (D) The source code is in the public domain.
6. Source code need not be distributed with the final product. Which of the following could be used?
   (A) Interpretation.
   (B) Compilation.
   (C) Byte code or other intermediate code.
   (D) Either B and C.
7. A program requires the user to make a selection from a total of 20 items. The most appropriate screen element to use would be a:
   (A) series of check boxes.
   (B) series of radio buttons.
   (C) list box.
   (D) menu.
8. If the RAND function returns a decimal between 0 and less than 1 then RAND*21+4 will return a value from:
   (A) 0 to less than 25.
   (B) 4 to less than 21.
   (C) 4 to less than 25.
   (D) 0 to less than 4.
9. Dynamic link libraries (DLLs) are used to add functionality to high-level languages. Which of the following is true of DLLs?
   (A) DLLs must be distributed with the final software product.
   (B) DLLs are combined within the executable file by the linker.
   (C) DLLs are only required for interpreted programs.
   (D) Each DLL must be present on the user's machine for the software product to execute correctly.
10. Hardware requirements for new software products need to be specified. Why is this the case?
    (A) Software products are often designed for specific combinations of hardware configuration.
    (B) It is often not possible for software developers to predict the hardware environment on which their products will be installed.
    (C) Software solutions will operate more efficiently under particular hardware conditions.
    (D) All of the above.

11. Translation of source code into executable files is necessary for all software solutions. Explain the processes involved in this translation process.

12. The fetch-execute cycle takes place for every task completed by a computer. Explain the processing that takes place to execute a simple high-level language assignment statement such as `Count:=1`. Answer in terms of the operations that take place within the CPU including references to the fetch-execute cycle.

13. Program development techniques aim to produce efficient and error free software solutions. Describe at least five techniques that are available to software developers that assist them in this task.

14. Select a new input device such as a recent game controller, scanning device, biometric device or touch device. Research how the device operates and how to collect data from the device within source code.

15. New technologies are constantly appearing. Using the internet, explore the most recent advances in CPU technology. Compare the speed and processing capabilities of these new processors with those available ten years ago. Do you think processing speeds and capabilities will continue to evolve at such a large pace?

In this chapter you will learn to:

- differentiate between systems and program test data
- test your solution with the test data created at the design stage, comparing actual with expected output
- use drivers and/or stubs to test specific modules and subroutines before the rest of the code is developed
- recognise the importance of module testing before the module or subroutine is incorporated into the larger solution
- recognise that while an individual program or module may have been successfully tested, when it is incorporated into a larger system, problems may become apparent
- demonstrate the features of a new system to the client
- assess the new software solution to ensure that it meets the specified quality assurance criteria
- assess the performance of the new software solution against the criteria specified by the benchmark

Which will make you more able to:

- identify and evaluate legal, social and ethical issues in a number of contexts
- construct software solutions that address legal, social and ethical issues
- apply appropriate development methods to solve software problems
- apply a modular approach to implement well-structured software solutions and evaluate their effectiveness
- apply project management techniques to maximise the productivity of the software development
- create and justify the need for the various types of documentation required for a software solution
- select and apply appropriate software to facilitate the design and development of software solutions
- assess the skills required in the software development cycle
- communicate the processes involved in a software solution to an inexperienced user
- use and describe a collaborative approach during the software development cycle
- develop and evaluate effective user interfaces, in consultation with appropriate people.

In this chapter you will learn about:

Testing the software solution

- comparison of the solution with the design specifications
- generating relevant test data for complex solutions
- comparison of actual with expected output
- levels of testing
  - module
    - test that each module and subroutine functions correctly
    - use of drivers
  - program
    - test that the overall program (including incorporated modules and subroutines) functions correctly
  - system
    - test that the overall system (including all programs in the suite) functions correctly, including the interfaces between programs.
    - acceptance testing
- the use of live test data to tensure that the testing environment accurately reflects the expected environment in which the new system will operate
  - large file sizes
  - mix of transaction types
  - response times
  - volume of data (load testing)
  - effect of the new system on the existing systems in the environment into which it will be installed

Reporting on the testing process

- documentation of the test data and output produced
  - use of CASE tools
- communication with those for whom the solution has been developed, including:
  - test results
  - comparison with the original design specifications

Evaluating the software solution

- verifying the requirements have been met appropriately
- quality assurance

Post implementation review

- facilitation of open discussion and evaluation with the client
- client sign off process

6

TESTING AND EVALUATING SOFTWARE SOLUTIONS

Testing and evaluation is integral to all stages of the software development cycle. In this chapter, we focus on higher levels of testing that take place after the project has been implemented in a programming language. Personnel within the software development process perform alpha testing with real data. Beta testing occurs when the product is distributed for use to a limited number of outside users. These users are engaged to report any faults or recommendations back to the software development company. In this chapter, we are concerned with both alpha and beta testing.



Alpha testing
Testing of the final solution by personnel within the software development company prior to the product's release.

The testing and evaluation process is central to a software development company's quality assurance. Our major aim is to ensure the product meets the original design specifications created during the 'Defining and Understanding the Problem' stage. In terms of quality assurance we also evaluate the success of the design and development processes. Another purpose of testing is to evaluate the product's performance against recognised industry standards or benchmarks.



Beta testing
Testing of the final solution by a limited number of users outside the software development company using real world data and conditions.

We discuss levels of testing, from unit or module level testing, through program testing and finally system testing. The generation of test data for specific purposes is discussed. Live test data is produced to test large file sizes, different transaction types, response times, volume data and also, of course, to ensure processing is correct. We examine CASE tools used to automate and report on the testing process. The results of the testing process are used to provide feedback so problems encountered can be corrected before the product is finally released.

We have already discussed many methods of testing in chapters 3, 4 and 5. Desk checks ensure algorithms operate as intended. Analysing the complexity of algorithms allows us to evaluate their efficiency. Test driven development requires tests to be designed before coding commences. Many of the debugging tools and techniques discussed in chapter 5 are used to test the code for errors. Test data together with expected results allows us to continually check our subroutines, modules and programs continue to operate correctly.

COMPARISON OF THE SOLUTION WITH THE DESIGN SPECIFICATIONS

The original specifications and requirements, which may have been modified during the development of the product, must be tested for compliance. The aim of the project is to implement the full set of design specifications. Because the requirements are written in such a way that they can be readily tested, a pass or fail can be allocated to each requirement. A checklist of all requirements is used. Each requirement is tested in turn. If faults are encountered then that aspect of the product is altered until the requirement is met.

Design specifications were developed as a framework for the development process. Each of these specifications needs to be tested against the final product. Many of the design specifications will involve internal issues; the results of the testing process will uncover inconsistencies in approach that have occurred during development. These inconsistencies must be identified and rectified both in the existing product and for future products.

HOW DO WE TEST THE DESIGN SPECIFICATIONS?

The remainder of this chapter will address this central question. During the testing and evaluation stage we aim to identify any errors, omissions and other problems within the product. Some problems may not be errors rather they maybe usability or hardware compatibility issues. Each problem identified will result in further development to correct the problem.

There are essentially two techniques used to test software solutions: white box testing and black box testing. Both techniques are required to identify and correct the majority of errors. Black box testing does not attempt to identify the source of the problem, but rather to identify that a problem exists. Once the problem has been identified, it may be necessary to move to white box testing to find and correct the problem. At this testing and evaluation stage, we are primarily concerned with black box testing techniques. The testing techniques used in previous chapters were white box testing techniques that aimed to identify and to correct problems emerging during the product's development.



Black box testing
Also known as functional testing. The inputs and expected outputs are known; the processes occurring are unknown.



White box testing
Also known as structural, or open box testing. A software testing technique whereby explicit knowledge of the internal workings of the item being tested is used.



GROUP TASK Discussion
Compile a list of testing techniques from both the 'Planning and Designing' and 'Implementation' stages of the Software Development Cycle. Explain why these techniques can be viewed as white box testing techniques.

LEVELS OF TESTING

Software products can be tested at various levels. Each individual subroutine and module within a program is tested as it is created. Each program will be tested to ensure modules work together correctly. We have examined both these levels of testing as part of the Implementation stage in chapter 5 (see section on Program Development Techniques). During the implementation stage, we were interested in the operation and processing of the actual programming code for the particular product. We now need to consider broader system-level testing.

WHAT IS SYSTEM-LEVEL TESTING?

System-level testing aims to ensure that the hardware, software, data, personnel and procedures that form the components of the final system are able to work together efficiently, correctly and in the manner intended with the new software product. Not only does the software need to operate correctly, it must operate correctly within the various system environments in which it will be installed and used.

To achieve this aim requires the use of real test environments utilising live test data. Let us examine each of the system components in terms of creating suitable test environments.

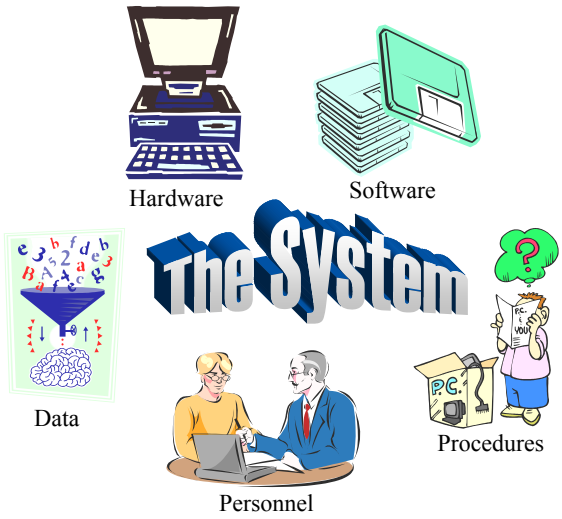


Fig 6.1
Testing aims to ensure all components of the system work with the new software product.

Hardware

The software product should be installed and tested on different combinations of hardware ranging from those specified as the minimum requirements to those that include any additional hardware devices.

Large systems are often designed for particular hardware configurations. This allows the product to be optimised for these specific machines. In this case, it is possible to test all possible hardware configurations. When testing products designed for a broad hardware base, it is common to encounter problems that are beyond the control of the software developer. For example, there are an almost countless number of different model printers available. It is not possible for software developers to guarantee their product will work correctly with all possible printers. Even if testing all printers were possible, any errors encountered are beyond the boundary of software products, and hence the software developer is often unable to solve the problem.



GROUP TASK Discussion
The office area of your school contains hardware, software, data, personnel and procedures. Describe the components of your school's office in terms of these five components.

Software

Other software installed on the system will have an effect on the operation of new software products. This is particularly the case when COTS products have been used during development. Different versions of the base product can cause incompatibilities with the new product. Similar problems can occur with different versions of the intended operating system. It is important to test the new product with various combinations of software installed.

New products that are designed to interface with existing products will require special attention. Often data is passed between applications and across networks. The only way to confidently test the interfaces between these applications is using live test data. Internet and web based applications require special attention as they often need to operate across a broad range of operating systems, variable network speeds and many different web browsers.

Data

The data input into the system is the most likely source of errors. The product needs to be tested with a large combination and quantity of live test data to ensure its reliability under live conditions. The section that follows deals with types of live test data necessary to ensure the robustness of the product when confronted with various live data scenarios.

Personnel and Procedures

The users of the system will have various skill and knowledge levels. Does the product cater effectively to the needs of users of the product? Particular attention needs to be given to the evaluation of the user interface. Beta testing is one of the most effective methods of testing requirements related to the personnel who will use the system. It is vital to choose a variety of different personnel to participate in the beta testing program.

The introduction of most new software products will involve changes in the way tasks are completed. Beta testing can be used to analyse the ability of personnel to efficiently implement any new procedures and to modify old procedures. Normally users are asked to comment on the effectiveness of any new procedures resulting from the introduction of the new software product.



Consider the following:

A new website has been developed by an office supply company. The company traditionally sells most of its goods by mail order. The new website allows purchases to be made using the website. The product is now at the beta testing stage.



GROUP TASK Discussion

Describe the type of businesses the office supply company may choose to be part of their beta testing program. Compile a questionnaire that could be used to evaluate the success of the new website in terms of issues related to personnel and procedures.

USE OF LIVE TEST DATA FOR SYSTEM TESTING

When a system is finally installed and implemented within the total system it is said to be 'live'. Once a product goes live it needs to undergo a series of tests to ensure its robustness under real conditions. This testing occurs using live test data.

Live test data is produced to simulate extreme conditions that could conceivably occur. The use of live test data aims to ensure the software product will achieve its requirements under these extreme conditions. Different sets of live test data are used to test particular scenarios. CASE tools are available to assist in the task of producing live test data sets and also to automate the testing process. Later in this chapter, we examine CASE tools used for this purpose.

For most products live test data should be created to test each of the following conditions:

- Large file sizes
- Mix of transaction types
- Response times
- Volume data (load testing)
- Interfaces between modules and programs
- Comparison of actual with expected results

LARGE FILE SIZES

Many commercial applications obtain input from large databases. During the development of software products, relatively small data sets are used. At the alpha and beta testing stages large files should be used to test the system's performance.

The use of large files will highlight problems associated with data access. Often systems that perform at acceptable speeds with small data files become unacceptably slow when large files are accessed. This is particularly the case when data is accessed via networks.

Large data files highlight aspects of the code that are inefficient or require extensive processing. Many large systems will postpone intensive processing activities until times when system resources are available. For example, updating of bank account transactions takes place during the night when processing resources are available.



Consider the following:

A rental car company is implementing a new computer system across its national network of 1200 branches. The branches are linked via a network of computers. Because of the remoteness of many branches, communication links are unreliable hence data must be held locally at each branch. Because cars can be returned to any branch, each branch requires a record of all rentals from all branches.

A system known as data replication is used. Essentially, every branch has a copy of the entire database. Changes to a particular record at one branch are duplicated across all branches every 10 minutes. The rental car company has a fleet of some 7000 vehicles with approximately 4000 new rental transactions occurring each day.

The software company developing this product wishes to create a large data file to use during alpha testing. Initially they create a file containing 400,000 transactions. The data replication is found to take on average 5 seconds when using this file.



GROUP TASK Discussion
Each rental transaction is held for five years. How many transactions should be in the test file to ensure performance remains satisfactory?



GROUP TASK Discussion
What types of problems are likely to be highlighted as a result of testing the product with such a large file? Explain.

MIX OF TRANSACTION TYPES

Testing needs to include random mixes of transaction types. Module and program testing usually involve testing specific transactions or processes one at a time. During system testing we need to test that transactions occurring in random order do not create problems.

Often the results of one process will influence a number of other processes. If a transaction is currently being completed on specific data and that data is altered by another transaction then problems can occur. When a number of applications are used to access the same database then conflicts are inevitable. Software must include mechanisms to deal with such eventualities.



Consider the following:

Banking systems involve a multitude of applications that access individual account details. Some accounts may experience many thousands of transactions per hour. This is particularly the case with public bodies such as the tax office and social security. Deposits and withdrawals are taking place at an enormous pace. These transactions originate from a large variety of sources: ATMs, bank branches, post offices, the Internet, etc.

New applications that interface with existing banking systems need to be able to deal with this situation. Transactions must be absolutely secure, given the nature of the application.



GROUP TASK Discussion
Describe possible scenarios that could result in problems when multiple transactions attempt to access a single account's details.



GROUP TASK Discussion
What type of live test data could be used to test the correctness of these banking transactions?

RESPONSE TIMES

The response time is the time taken for a process to complete. The process could be user activated or it could be activated internally by the application. Response times are dependent on all the system components, together with their interactions with each other and other processes that may be occurring concurrently.

Any processes that are likely to take more than one second should provide feedback to the user. Progress bars are a common way of providing this feedback. Data entry forms that need to validate data before continuing should be able to do so in less than one second; 0.1 second is preferable. Studies have shown that users will tolerate delays of up to one second, however their concentration will deteriorate if the delays are frequent. Response times of around 0.1 second give the user the impression of instantaneous response; this is of course the ideal situation.

Response times should be tested on minimum hardware using typical data of different types. Any applications that affect and/or interface with the new product should be operating under normal or heavy loads when the testing takes place.



Consider the following:

On the World Wide Web, response time rules! Sites that include animations and complex graphic advertisements often reduce the effectiveness of the site by increasing response times. The time taken to download these graphics can often cause prospective customers to give up and go to a faster competitor's site. There is money to be made by closely examining the response times for each transaction on a new website.

There are many companies that will analyse the response time for different aspects of websites. Their findings can result in significant savings for their clients. One company was founded by an individual who advertised his business with the slogan 'I'll tell you why your website is rubbish'. His business was based on response time analysis.



GROUP TASK Discussion
Explain why response times on websites are so crucial to the success of these sites.



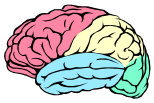
GROUP TASK Discussion
Describe the nature of tests that could be implemented to test the response times for websites. Justify your answers.

VOLUME DATA (LOAD TESTING)

Large amounts of data should be entered into the new system to test the application under extreme load conditions. Multi-user products should be tested with large numbers of users entering and processing data simultaneously. Large systems that require extensive data input require special consideration.

Alpha testing by the software developer must try to simulate the number of users who will simultaneously use the product. This can be done using a laboratory of computers and users in conjunction with CASE tools. In many cases, it is difficult to simulate real volumes of data in a beta testing environment without actually implementing the system.

CASE tools are available that enable automatic completion of data entry forms. Many of these tools allow the creation of virtual users. This allows one machine to simulate the activities of hundreds or even thousands of simultaneous users entering data. We examine CASE tools in detail in the next section.



Consider the following:

The post office is implementing a new application in its existing computer system. This application allows businesses to purchase and print stamps via the Internet. It is envisaged that some 20,000 businesses will take advantage of this initiative within one year of its introduction. Within five years, some 150,000 businesses will be using the service.

The software has been written and is about to undergo volume testing. Obviously it is not possible to beta test the product with 150,000 users, so a simulation using CASE tools is used.



GROUP TASK Discussion
The volume of data to be processed is crucial to the success of this product. How can volume testing be undertaken when the customer base does not at present exist?



GROUP TASK Discussion
The security of these transactions is crucial. Discuss methods that could be used to test the security of the system, both in terms of payment for stamps and also in terms of ensuring printed stamps are not counterfeit.

INTERFACES BETWEEN MODULES AND PROGRAMS

An interface provides a communication link between system components. In terms of modules within a program, the interface is usually provided through the use of parameters that are used to pass data to and from modules. The screens of an application provide an interface between the users and the program. Hardware drivers are programs that provide an interface between applications and hardware devices. Programs that interact with other programs require an interface to complete their tasks. All these interfaces require testing.

In this section, we are interested in testing the performance of interfaces that connect modules and programs. Tests and test data need to be created to analyse the interaction between different software components of the system. In many large systems the interface between programs will include a communication link, this communication link being an integral part of the interface. Tests should examine the accuracy of data being passed as well as response times and the ability of the interface to cope with large files, different transaction types and large volumes of data.



Consider the following:

An upgrade of the software that operates a particular type of Automatic Teller Machine (ATM) is being alpha tested. The system flowchart in *Fig 6.2* describes the programs with which this application interfaces.

The ATM application must wait for authorisation from the main bank application. The bank application sends an enquiry to either its own database or to another financial institution. It must then await a response. The other financial institution must then perform an enquiry and wait for a response before sending the result to the originating bank. The originating bank then sends a response to the ATM application.

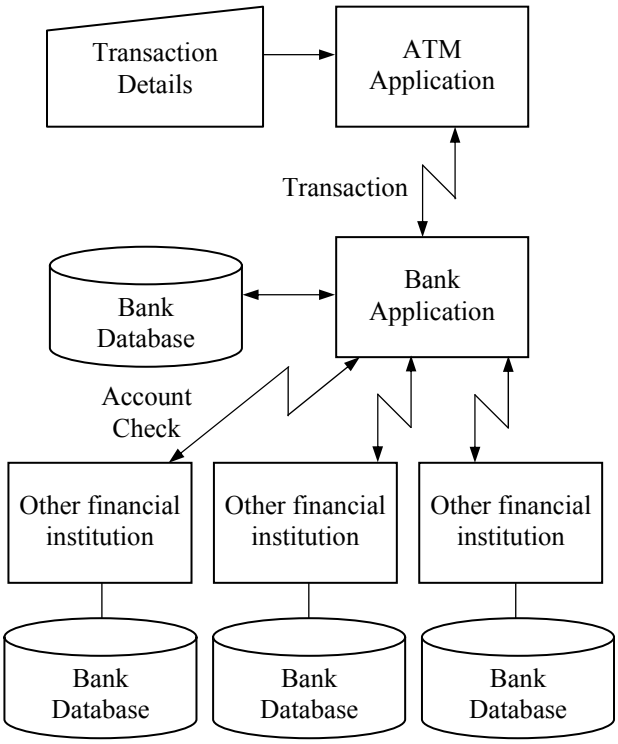


Fig 6.2
System flowchart for ATM interfaces.

Tests designed for testing this system must examine any eventuality. Perhaps a data link is down, or an application is slow to respond. The test data must ensure that the ATM application responds appropriately to all occurrences.



GROUP TASK Discussion

Develop a list of possible scenarios that must be tested to ensure the correct operation of the ATM application in terms of interfaces to other applications. For each scenario describe the type of test and test data that could be used to ensure correct operation of the ATM.

COMPARISON OF ACTUAL WITH EXPECTED RESULTS

During previous stages of the software development cycle, test data together with expected outputs were created. This test data should be entered into the finished product to ensure the actual outputs match the expected outputs. Because the correct outputs are already known, this testing process ensures the final product is able to perform its processing correctly.



GROUP TASK Discussion

Program test data has already been entered and the results analysed during earlier stages of the software's development. Why is it necessary to input and check the results again? Discuss.

BENCHMARKING



Benchmark

A point of reference from which quality or excellence is measured.

Benchmarking is the process of evaluating a product, in this case a software product, using a series of tests. These tests provide numerical results that are compared with recognised standards. The process of benchmarking aims to compare the performance of a product with these standards. Results of benchmark tests are used to compare products. These results allow users of software products to make informed purchasing decisions. They also provide software developers with objective data, which can be used to compare their products with those of their competitors.

There are many variables that affect the performance of a software product. A particular product may outperform its competitor on a particular hardware configuration, however the results may be reversed on a different configuration. Benchmarking aims to reduce the number of variables and to report results objectively.

Successfully completing benchmark tests requires copies of competitors' products. Both products are subjected to a series of tests using identical hardware configurations and test data sets. The results are used to evaluate the performance of the product compared to its competitors. Benchmarking a product is vital to maintaining and improving a product's market penetration.

Although benchmarking is often undertaken internally by software development companies without the cooperation of their competitors, this is not always the case. It is often in the interest of both companies to work together on benchmarking activities. Cooperation between software companies can result in mutual gains and an increase in quality and performance across the industry.

Companies specialising in benchmarking are emerging. These companies provide benchmarking software that can be utilised to test specific aspects of computer systems. Industry associations, as a result of their unbiased nature, often create and distribute benchmarking products. Most of these products test the performance of specific hardware functions when subjected to particular software commands: for example, graphics rendering, file input and output, or floating-point calculations. Results from these benchmarking tests can be used to optimise software products for particular system configurations.



Consider the following:

A software development company has developed a new product to allow businesses to fax directly from their computer to multiple recipients. The details of fax recipients can be extracted from a number of common database formats.

There are two other products on the market that perform similar functions. The software development company wishes to evaluate their new product by comparing it with their competitors' products.



Fig 6.3
Software products are available to allow faxes to be sent directly from a computer.



GROUP TASK Discussion

Make up a list of tests that could be employed to assist in the benchmarking process. Explain why each test has been included.



GROUP TASK Discussion

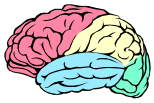
The system on which the tests take place is vital to obtaining reliable results. What criteria could be used to assist in the selection of appropriate system components for these tests? Discuss.

QUALITY ASSURANCE

The quality of a software product is measured against how well the product meets or exceeds users' expectations. Quality assurance is about ensuring that this occurs. In chapter 1, we examined aspects of quality software. Each of these aspects requires testing and evaluation. The following factors and related questions are reproduced from chapter 1:

- Correctness: Does it do what it is supposed to do?
- Reliability: Does it do it all of the time?
- Efficiency: Does it do it the best way possible?
- Integrity: Is it secure?
- Useability: Is it designed for the end user?
- Maintainability: Can it be understood?
- Flexibility: Can it be modified?
- Testability: Can it be tested?
- Portability: Will it work with other hardware?
- Re-useability: Can parts of it be used in another project?
- Interoperability: Will it work with other software?

Quality assurance is not just about testing a product once it has been completed. Quality assurance techniques should be implemented throughout the software development process as well as throughout the processes of the software development company. National and international standards exist for quality assurance. The International Standards Organisation (ISO) develops standards for quality assurance. Companies and products can adopt these standards. Organisations throughout the world provide certification services to businesses wishing to participate. In Australia, Quality Assurance Services (QAS) is one of the largest suppliers of such services. QAS provides initial certification services, product testing, and certification, together with auditing and education services. Products and companies are able to use standards logos on their advertising and product packaging.



Consider the following:

A software quality assurance company is employed to test a software application that has been written to control the operation of a digital video camera. The software company is particularly interested in results pertaining to correctness, reliability, efficiency and useability.

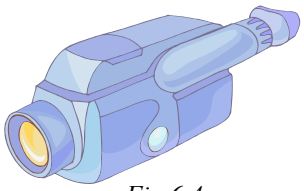


Fig 6.4
Digital video cameras are controlled by software applications.

The camera performs all the usual video camera functions together with some new and innovative functions. For example, the camera can be set to commence filming once it detects movement. The camera will then follow the source of the movement, maintaining the subject within each frame. This function is particularly useful for wildlife photographers who traditionally spent hours waiting to capture particular wildlife footage.



GROUP TASK Discussion

Describe the types of tests that could be conducted by the quality assurance company to determine the correctness, reliability, efficiency and useability of this product.



GROUP TASK Discussion

Results of quality assurance are often used to assist in the marketing of products. How could the results, assuming they are positive, be used to market this new video camera?

SET 6A

1. Internal system testing using live test data can be best described as:
   (A) white box testing.
   (B) black box testing.
   (C) alpha testing.
   (D) beta testing.

2. The main purpose of the testing and evaluation stage of the software development cycle is to:
   (A) discover bugs in the source code, at both the module and program level.
   (B) ensure the program operates correctly with live test data.
   (C) train users in the operation of the new product.
   (D) ensure the product meets its requirements and design specifications.

3. Explicit knowledge of the internal workings of an item is utilised when performing:
   (A) white box testing.
   (B) black box testing.
   (C) alpha testing.
   (D) beta testing.

4. System-level testing should include all components of the system. This includes:
   (A) modules, programs and systems.
   (B) hardware, software, data, personnel and procedures.
   (C) large file sizes, mix of transaction types, response times, and volume data.
   (D) benchmarking and quality assurance.

5. Benchmarking is:
   (A) used to ensure the product is of a high quality.
   (B) a technique used to isolate and report weaknesses in the product.
   (C) used to compare a product to other, similar products using a standard point of reference.
   (D) intended to gauge the extent to which a product has realised its goals and objectives.

6. Users testing a software product before final release are participating in:
   (A) white box testing.
   (B) black box testing.
   (C) alpha testing.
   (D) beta testing.

7. The inputs and expected outputs are known, however the details of the processes that transform inputs into outputs are unknown during:
   (A) white box testing.
   (B) black box testing.
   (C) alpha testing.
   (D) beta testing.

8. Large files are likely to highlight problems in regard to:
   (A) errors in the logic of the source code.
   (B) inefficient or processor intensive operations.
   (C) the operation of the user interface.
   (D) All of the above.

9. Tests in regard to response times are formulated to primarily test the:
   (A) operation of the user interface.
   (B) hardware components of the system.
   (C) procedures required when operating the new product.
   (D) product compared to competitors' products.

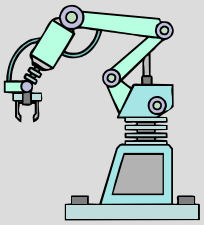
10. Which of the following is NOT true in regard to Quality Assurance (QA)?
   (A) QA practices should be implemented throughout the development process.
   (B) QA is primarily about ensuring customers' expectations are realised in the final product.
   (C) QA ensures testing of software products results in error-free applications.
   (D) QA standards are available for implementation by businesses. Certification allows use of standards logos.

11. Black box testing is the major technique used during the testing and evaluation stage of the software development cycle. Why is this the case, and why is black box testing an appropriate technique? Justify your answer.

12. Live test data is used to test a number of aspects of the total system. Describe aspects of the system that are best tested using live test data. Why can't program test data be used to achieve the same result? Discuss with the use of examples.

13. Benchmarking a product against its competitors is a common technique. Software companies, media groups and industry organisations all complete benchmark testing. Describe the process of benchmarking. How can benchmarking assist in raising the standard and quality of software products?

A software product has been developed to control a robotic production line. The production line manufactures plastic automotive components. These components must be produced within particular size tolerances. The robots perform both the manufacturing and the measuring of each component prior to its acceptance.



Details of the plans for each component are obtained directly from a CAD (Computer Aided Design) file. Size tolerances are input into the new system via a graphical user interface.

14. The software development company has obtained a robot that is able to produce prototypes of individual components. This robot is not able to perform the size tolerance checks. Explain how this robot could be used during alpha testing. What techniques could be employed to check the size tolerance modules of the software product without the use of the correct robotic hardware?

15. Quality Assurance is vital to the motor vehicle manufacturers for whom the components are manufactured. What types of procedures and testing methods should be used to ensure the new software product meets required quality standards?

CASE TOOLS FOR TESTING SOFTWARE SOLUTIONS

A large variety of Computer Aided Software Engineering tools are available for use during the testing and evaluation stage. These tools automate the more tedious tasks associated with the testing process. Many tools specialise in particular testing functions such as test data generation, user interface testing, or volume testing.

Let us examine a number of CASE tools available at the time of writing:

WinRunner

WinRunner is a functional testing CASE tool. It aims to test that user interface-based applications behave as expected. This product automates the tasks associated with user data entry and navigation.

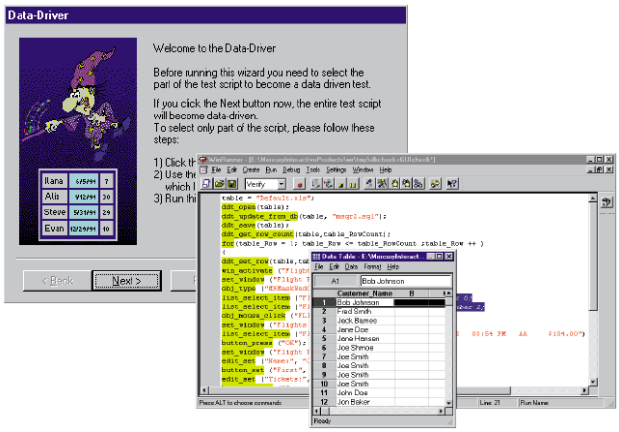


Fig 6.5
Sample screens from Mercury Interactive's WinRunner product.

WinRunner is able to test applications written in most popular languages. Applications written for mainframes can be tested using terminal emulation software.

The tester records the keystrokes necessary to complete a particular process. A script is created based on these recorded keystrokes. The script can then be edited. Data values are replaced with parameters: this allows the test to be executed with multiple test data items. Similarly, parameters can be included to cause outputs to be stored for future analysis. Checkpoints can be inserted into the script that cause WinRunner to compare expected results with actual results.

LoadRunner

LoadRunner is a volume or load testing CASE tool. This tool is able to simulate hundreds or even thousands of users simultaneously using an application.

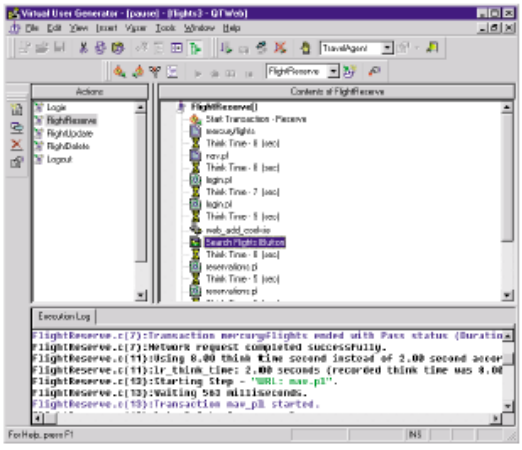


Fig 6.6
Sample screens from Mercury Interactive's LoadRunner product.

Scripts are recorded and then parameterised to allow the use of different live test data. Virtual users are created. Each virtual user can have different characteristics. For example, different time intervals can be set between an individual user's inputs. Tests can be designed that cause multiple virtual users to enter data into the same field simultaneously.

This product contains a number of real-time monitors allowing you to view the application's performance whilst the test is executing. For example, the transaction monitor provides information on average transaction response times, and the server monitor identifies problems associated with application and database servers.

DataTech

DataTech is a universal test data generation CASE tool. This product is able to create millions of test data records. Rules for data creation can be produced within the product or determined from existing databases. The test data sets can be written to a large variety of database formats or to text files.

Data can be randomly generated or can be extracted from other data sources. For example, address details such as cities and postcodes can be extracted from a database containing this information.

Data generated by DataTech can be used in conjunction with other testing CASE tools.

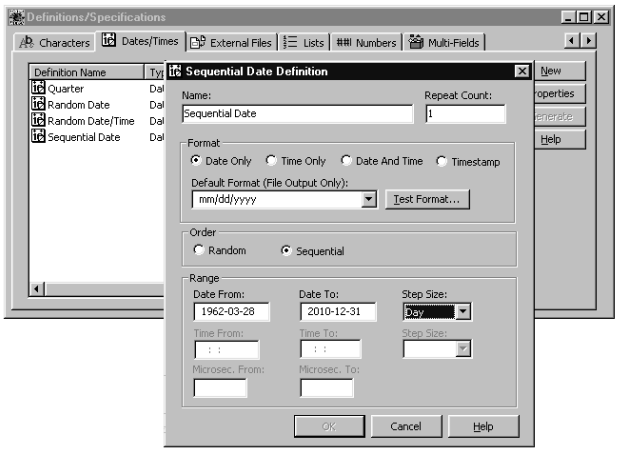


Fig 6.7
Test data generation tool DataTech by Banner Software Inc.

UsableNet

This is a website dedicated to testing the useability of other websites. The application analyses the HTML code behind a website and then generates a series of useability reports as well as making recommendations.

Reports are generated on each page of the tested website. Details of each potential problem are made available by clicking on each problem area.

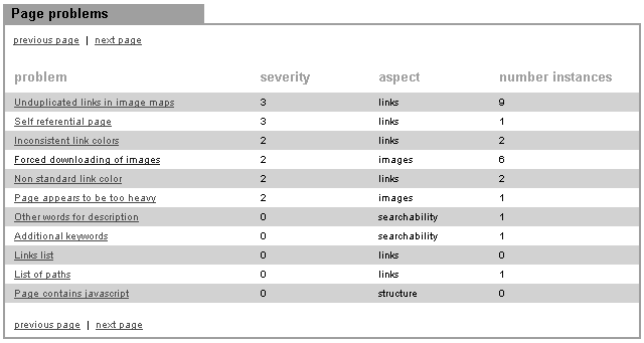


Fig 6.8
UsableNet provides a test service for websites.



GROUP TASK Research

Use the Internet to search for software testing tools. Collect data on the name of the product, its main function and a brief description of how it works. Create a table to summarise your findings.



GROUP TASK Activity

Download a demonstration version of a functional testing tool from the Internet. Use this tool to create and execute a test on a data entry screen for an application with which you are familiar.

REPORTING ON THE TESTING PROCESS

Documentation and communication of the testing process and its results are vital to any testing program. Results of testing need to be evaluated and acted upon if the problems highlighted are to be corrected. For this to successfully occur requires documentation of the testing process and communication of the results to the development team and to users.

DOCUMENTATION OF THE TESTING PROCESS

Documentation should be developed and retained as the testing process is undertaken. Documents produced will include the following:

- Test requirements — What needs to be tested?
- Test plan — How do we implement these tests?
- Test data and expected results — What are the necessary test inputs and the expected outputs?
- Test results — Do the actual results match the expected results?
- Recommendations — What needs to be done now?

Test requirements

The test requirements will reflect the requirements and design specifications for the project. Essentially, test requirements detail the scope of the testing process: What tests need to be undertaken, and why? Each component of the program and the system should be considered as part of the test requirements. Hardware, software, data, personnel and processes should all be tested.

Test plan

The test plan will include a schedule and time line of events that will occur during the testing process. It also includes the tools to be used, including software CASE tools and hardware.

Test data and expected results

The actual test data and expected results used throughout the testing process should be retained as documentation. The test data may be required again after modifications have been made to correct problems.

Test results

The test results should be kept to justify any recommendations for changes to the product. These results will be invaluable later to confirm that changes have in fact been implemented.

Recommendations

As a result of the testing process many recommendations will be made in regard to bugs and other problems encountered during the testing process. The recommendations should succinctly describe each problem and explain its severity. Management, in conjunction with the client, will decide which recommendations to act upon and which are acceptable problems. The development team is then handed the task of applying these accepted recommendations.

COMMUNICATION OF TEST RESULTS

The nature of the project will determine the audience for the test results. Projects written for a specific client will require communication of results directly to the client. Large software development companies usually have separate development and testing personnel. In this case, the testers must communicate their findings back to the development team. Often software developers outsource their system testing to specialists. If this is the case, the testing specialists must communicate their findings back to the software development company.

The test results will include problems from all aspects of the system tests. Bugs in the code, inconsistencies in user interface design, unacceptable response times from other applications, hardware conflicts and ambiguous error messages are just some of the possible problems.

Communication with the client for whom the solution has been developed

The language used to report back to users needs to be non-technical. Reporting should address any requirements and design specifications that have not been adequately fulfilled. The reporting process should be in terms of the client's original requirements.

It is normal to rank problems in terms of their severity: a crucial requirement that has not been fulfilled would rank highly; minor problems with the response time of a particular function would rank lower. Some problems encountered may be outside the scope of the project; these problems should be made known. It is always better to be upfront and honest about the capabilities of the product. After all, the client will be using the product regularly so eventually they will become aware of problems.

To assist in the communication of test results it is common to give a demonstration of the system to the users. During this demonstration both positive and negative features can be shown. Users are then able to discuss and evaluate the product more realistically.



Fig 6.9
Onsite practical demonstrations are an excellent method of communicating test results to clients.

Communication with developers

Testing departments or companies outsourced to complete the testing process need to communicate their results back to the development team. The communication should highlight problems in order of severity. The recommendations must be backed up by technical justifications. The actual test data together with the erroneous results will provide valuable information for the developers who will be correcting or modifying the product.

Personnel problems are common at this time. Developers often assume the product is finished; to be presented with a list of faults is often viewed personally, as an assault on their professionalism. Testers and developers need to encourage a climate of cooperation and teamwork. The quality of the final product should be the overriding concern.



Consider the following:



A pest control company called 'Erad-A-Pest' has some 50 trucks completing domestic pest control services. The company has employed a software development company to develop and implement an automated booking, scheduling and invoicing system. Each pest controller is notified of jobs via text messages sent over the Internet to the mobile phone digital network.

The software has just undergone alpha and beta testing. The following table lists a number of the problems discovered during the testing process.

Problem	Source	Description	Severity
Duplicate orders processed.	Orders module	If a client sends an order twice, this is not detected by the system.	High
Address validation slow.	Scheduling module	Client addresses are validated against their phone number. This takes between 2 and 3 seconds.	Low
Discounts calculated incorrectly.	Invoicing module	Discounts are deducted after GST has been calculated.	Extreme
Inconsistent acknowledgement of receipt of text messages.	Scheduling module	Acknowledgement is not received for approximately 5% of text messages sent to pest controllers.	High
System does not reconnect to ISP.	All modules	Loss of connection to ISP requires restarting the software.	Low
Completion of job notifications inconsistent.	Procedural	Pest controllers do not submit 7% of job completion notifications.	Medium



GROUP TASK Discussion

Examine each of the problems identified in the above table. Suggest strategies for correcting each of these problems.



GROUP TASK Activity

Create two written reports outlining your recommendations as a consequence of these test results. One report should be targeted at the users of the system and the other at the development team.



GROUP TASK Discussion

Which method of conversion is appropriate for the introduction of this product: pilot, parallel, direct cut over or phased? Justify your answer.

EVALUATING THE SOFTWARE SOLUTION

The major aim of testing is to detect and correct errors before the system is installed for use. At the system level we are largely concerned with assessing the extent to which all the requirements and design specifications have been met. This assessment has the added bonus of providing critical information with which to evaluate both the new system and also to evaluate the design and development process including the quality assurance procedures.

The original design specifications include requirements for the particular application together with specifications in regard to documentation required, screen design and code design. In essence, these specifications describe what the software should do together with how it should be done. Evaluation of the design specifications will therefore ensure the product realises its requirements and that those involved in the design process have maintained standards in regard to how the requirements have been achieved.

POST IMPLEMENTATION REVIEW

A review of the system once operational is often undertaken prior to the client formally accepting the software as complete. For smaller projects this evaluation is often in the form of discussion between the client and the installers. Commonly the discussion will include demonstrations to confirm and describe how the requirements have been met. For large systems professional independent assessors perform thorough and rigorous acceptance tests followed by a formal review. The acceptance tests and review confirm the requirements have been met prior to the client signing off on the project and paying all outstanding moneys.

ACCEPTANCE TESTING

Acceptance tests are designed to confirm the requirements have been met so the system can be signed off as complete. The client and the developers usually agree to use the results of the acceptance tests as the basis for determining completion of the new system. If the tests are successful then the client makes their final payment and the development team's job is complete.



Acceptance Tests
Formal tests conducted to verify whether or not a system meets its requirements. Acceptance testing enables the client to determine whether or not to accept the new system.

For large-scale information systems acceptance testing is best performed by an outside specialist testing organisation. Even for smaller systems it is preferable for acceptance tests to be performed by people who were not involved in the software's development. People involved in the development process are likely to be biased. They have designed and implemented the new system, so clearly they will feel the requirements have been met. Furthermore they will, unsurprisingly, view their particular solution as superior to other possibilities.

Although using outside testers is preferable, it is not unusual for the client to perform their own acceptance tests prior to finally accepting and signing off the new system. This is understandable, given that all systems are ultimately developed to meet the needs of clients. Unfortunately disagreement between the clients view of an acceptable system can differ from the views of the developers. It is preferable to agree on the precise nature of the testing and who will perform the tests early in the development cycle – in terms of the traditional development approach this should occur during the creation of the requirements and design specifications. This can easily become a significant problem with less structured development approaches.



HSC style question:

Question 1.

With the aid of examples, distinguish between program level testing and system level testing.

Question 2.

Write down THREE different types of live tests and describe the nature of the issues each type of live test aims to detect.

Suggested solutions

Question 1.

Program test data is designed to test the internal logic of the program. In particular it ensures all subroutines and modules work function together correctly. It should use carefully selected test data which tests calls to each subroutine and then tests every pathway and boundary point in the logic. Thus if a program includes a subroutine to determine if a person is of voting age with a date of birth as the input, the test data should ensure the routine is called with 3 items – a date of birth for a person aged less than 18, a date of birth for a person aged exactly 18 and a date of birth for a person aged over 18.

Systems test data is used to test how the overall system will perform and function when live. For example, to test a new banking system that will run with 40,000 customers, expecting 500 transactions per day of which almost half are withdrawals, another almost half are deposits, and perhaps 1 percent are transfers, test data reflecting this number and type of transactions is generated and used as a trial for input into the new system. System tests also include acceptance tests to ensure the final new system meets all the client's requirements.

Question 2.

- Large file sizes: Small data sets are often used during development, but it is important to test using large data sets and file sizes as this can highlight performance problems (particularly inefficient processing of large files) along with issues when using large file sizes across a computer network.
- Response times: The response times of programs should be tested on a variety of platforms, hardware specifications, and conditions (e.g. high memory usage, large number of other applications running in the background, light load, heavy load) to ensure that the program does not become unacceptably slow or unresponsive.
- Volume of data (load testing): The system should be tested under load with many simultaneous users entering large amounts of data as this will ensure that the application can cope with that amount of data from processing, storage, and performance perspectives. CASE tools can be used to simulate the volume data as this is often difficult to accomplish manually.

CHAPTER 6 REVIEW

1. Testing is carried out during which stage of the software development cycle?
   (A) Planning and design.
   (B) Implementation.
   (C) Testing and evaluation.
   (D) All of the above.

2. Problems encountered during system testing are often the result of hardware problems. Which of the following is true?
   (A) Correcting problems due to hardware is beyond the scope of the software developer.
   (B) The problem is usually solved by modifying the hardware requirements of the software.
   (C) Correcting faults in hardware is generally not possible; therefore software developers need to find software solutions to allow their products to operate correctly.
   (D) Hardware faults are inevitable, as are software faults. Users generally accept this as a fact of life.

3. CASE tools used during the testing evaluation stage:
   (A) are used to automate tedious and repetitious tasks.
   (B) provide a means of producing system models.
   (C) assist developers in the construction of code.
   (D) ensure design specifications remain consistent across the application.

4. Customised off-the-shelf (COTS) packages can greatly simplify and shorten the software development process. Testing of software based on COTS packages can be difficult because:
   (A) differing versions of the COTS package may cause changes and/or problems to the product.
   (B) errors in the COTS package cannot generally be rectified by the software developer.
   (C) other products that use the services of the COTS package can affect functionality in unexpected ways.
   (D) All of the above.

5. The difference between live test data designed to test large file sizes and that designed to test volume data is:
   (A) large file sizes test the ability of the software to retrieve data; volume data tests the ability of the software to process large amounts of input simultaneously.
   (B) large file sizes test the ability of the software to process data correctly; volume data tests the ability of the software to process large amounts of data.
   (C) large files ensure the product is robust in design; volume data ensures logic errors are identified.
   (D) large file sizes can cause arithmetic overflow errors which must be identified; volume data is used to test the user interface.

6. Why should program test data be used again during system testing?
   (A) It is easier than producing new test data sets.
   (B) Problems identified during system testing are the same as those identified during program testing.
   (C) The outputs from both tests can be compared. Differences will highlight system issues compared to program issues.
   (D) The expected outputs are known, hence correcting any errors will be straightforward.

7. Live test data:
   (A) ensures the logic of source code is error free.
   (B) is used to test the operation of the system under extreme conditions.
   (C) is only used during beta testing.
   (D) can be used most effectively once the program has been distributed.

8. CASE tools that can simulate the activities of a large number of users are used for:
   (A) functional testing.
   (B) load testing.
   (C) test data generation.
   (D) communicating test results.

9. Documentation should be created throughout the testing process. The major purpose of this documentation is to:
   (A) justify the costs of the testing process.
   (B) provide evidence to support the recommendations.
   (C) involve testers in the development of the product.
   (D) ensure that the integrity of the testing process is maintained.

10. Communicating test results effectively is crucial to the testing operation. Why?
   (A) This is the means by which recommendations will be put into practice.
   (B) The significance of problems must be made clear.
   (C) Problems not corrected now will prove more costly to correct once the product has been distributed.
   (D) All of the above.

11. The user interface is a vital component of most data-oriented software products. The layout and nature of screen elements together with response times and data validation processes all influence the experience for the user. What testing techniques can be used to test the user interface? Describe how live test data can be used to assist this process.

12. Live test data is commonly used to perform system testing. Different test data sets are designed to test particular aspects of the final product. Describe at least three different scenarios that require different test data sets. How can each of these test data sets be obtained or created?

Use the following scenario to answer Questions 13 and 14.

A watch which is controlled by a microprocessor is under development. This watch combines the functions of a traditional watch with a number of innovative features designed specifically for use by athletes. Heart rate, blood pressure and body temperature are constantly monitored. A low and high value for each of these three values can be input using buttons on the side of the watch to scroll through possible values. Readings that are outside the input range cause an audible alarm to be emitted that alerts the wearer. The software that controls the watch is now ready for alpha testing, together with a prototype of the final watchcase and associated hardware.



13. What is alpha testing? Develop a list of tests that should be performed on the watch during the alpha testing process. For each test describe the nature of any test data required.

14. A number of faults are identified during the alpha testing. The following list describes these faults:
   - Watch strap needs to be tight for temperature sensor to operate reliably.
   - Once a range is set for the heart rate function, removing the watch causes the alarm to sound.
   - Body temperatures reported by the watch are wrong. Many temperatures reported are outside those possible for a living person.
   - The date does not take account of leap years correctly.

   It seems strange to the testers that many of these faults were not encountered during the coding of the software. Can you explain this situation? Discuss in relation to each fault.

15. Imagine you have been enlisted to beta test a new software product. A large multi-national software company has developed this product. The product is called 'Smart Type'. Essentially, it attempts to guess the word you are typing, the aim being to reduce the time spent entering data into word processors. Guesses are based on past words you have entered, the theory being that it is likely you will want to use these words often. The product can, apparently, be used with any existing word processor.

   Develop a series of tests you would use to beta test this product. Describe the purpose of each test you devise. Your response should include tests of the hardware, software, data, personnel and procedures involved in the use of this software product.

In this chapter you will learn to:

- read and interpret source code created by other developers
- design, implement and test modifications
- recognise the cyclical approach to maintenance
- document modifications with dates and reasons for change

Which will make you more able to:

- differentiate between various methods used to construct software solutions
- identify and evaluate legal, social and ethical issues in a number of contexts
- construct software solutions that address legal, social and ethical issues
- apply appropriate development methods to solve software problems
- apply a modular approach to implement well-structured software solutions and evaluate their effectiveness
- apply project management techniques to maximise the productivity of the software development
- create and justify the need for the various types of documentation required for a software solution
- select and apply appropriate software to facilitate the design and development of software solutions
- assess the skills required in the software development cycle
- communicate the processes involved in a software solution to an inexperienced user
- use and describe a collaborative approach during the software development cycle
- develop and evaluate effective user interfaces, in consultation with appropriate people

In this chapter you will learn about:

Modification of code to meet changed requirements

- identification of the reasons for change in source code
- locating of sections to be altered
- determining changes to be made
- implementing and testing solution

Documenting changes

- including relevant comments in the source code to highlight the modification
- updating associated hard copy documentation and online help
- using CASE tools to monitor changes and versions

7

MAINTAINING SOFTWARE SOLUTIONS

Maintenance is an ongoing process of correction and refinement. Many software products have a life span of some ten to fifteen years. These products will require maintenance for them to continue to meet the expectations of their users. Maintenance includes the correction of errors (bugs) in the source code together with upgrades to enhance the functionality of the product to meet new or changing requirements.

In this chapter, we examine the processes involved in maintaining a software solution. We examine methods of identifying and determining what changes should be made. This process involves ongoing communication with the product's users. Once a decision has been made to modify the product, the software development cycle is commenced. Finally, a method of distributing the modification is devised, and the modification is installed for use. For example, small bug fixes may be distributed as a patch, whereas large changes to the user interface may require a complete upgrade to be distributed.



Patch
Used to correct a problem in a software solution. A patch is an actual piece of executable code that is inserted into (patched into) an existing executable program.

Documentation of the original product's development will prove invaluable to maintenance programmers. Maintainability is a measure of how easily a solution can be maintained; it is closely linked to the quality of the original documentation. Often the maintenance programmer will not be the same as the original programmers. Many software development companies have a separate team of maintenance programmers. The task of maintaining a product is simplified when the original technical documentation is thorough and consistent.



Consider the following:

Today many large software development companies, together with their distributors, maintain websites where customers can report bugs and other problems found in their products. Often these sites include a series of bug fixes, in the form of patches that can be downloaded.



GROUP TASK Investigation
Visit the websites of a number of large software companies. Make a note of any software maintenance features that exist on each of these sites. Why would a software company choose to include bug fixes on their site?

MODIFICATION OF CODE TO MEET CHANGED REQUIREMENTS

Software solutions are not static entities - they evolve over time. Most applications mature as new versions are introduced. These new versions will correct bugs and introduce new functionality not present in the original product. Changes made to subsequent releases of applications are often the result of user requests. Many software companies provide facilities for users to make suggestions in regard to new functionality or changes to existing functionality. Remember, fulfilling the requirements of end users is a central aim for most software products.

IDENTIFICATION OF THE REASONS FOR CHANGE IN SOURCE CODE

The possible reasons for changing source code are many and varied. Reasons for change common to all applications include: bug fixes, new hardware and new software purchases. Major upgrades of operating systems will have a flow-on effect to other applications. Upgrading of applications to take advantage of features available in new operating systems is required.

A software company whose products have a large customer base may receive many thousands of requests for modification of their products. The company must then prioritise these requests and make decisions on which modifications will be included and which will not. Often registered users are surveyed to obtain their views on possible modifications. Support departments provide a valuable source of data: the number of support issues about specific functions highlights useability problems. This information can be used as the basis for selecting appropriate modifications for inclusion in an upgrade.

Customised software written for a specific client will require continual maintenance as requirements change. The focus of companies change, small companies will grow and their procedures will change. As customised software is an expensive purchase, maintaining the product is normally preferable to changing products, or developing a new product from the ground up. Because custom software is written to solve a particular problem, minor changes to the requirements often necessitate alterations and additions to the source code.

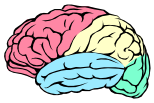


Script
A list of commands that can be executed without user interaction. A script language is a simple programming language with which you can write scripts.

Software based on COTS products is particularly susceptible to upgrades in the base COTS product. Macro and script commands may change, resulting in unexpected results. Often applications will need to be recompiled using the new version of the COTS product. The Internet has automated the upgrading of many commonly used applications. Products using the resources of these applications may require similar upgrading to continue to operate as intended.



Macro
A symbol, name, or key combination that activates a series of commands, actions, or keystrokes. Many programs allow the creation of macros to automate common tasks.



Consider the following:

A multimedia title uses Microsoft's Media Player to play video sequences within the product. Microsoft has just released a new version of Media Player that incorporates a more graphically rich set of user controls. The multimedia title, although still operable, does not contain sufficient space for the new Media Player interface. Many end-users have contacted the software company in regard to obtaining an upgrade to correct this problem.



GROUP TASK Discussion

'Microsoft should not make changes to products that impact other products. In this case they have done so, therefore they should be responsible for any costs incurred in modifying the multimedia title.' Do you agree with the above statement? Discuss.



GROUP TASK Discussion

The author of the multimedia title decides to correct the problem and sends out a patch to all registered users. How could they correct the problem so that future upgrades to Media Player will not affect the operation of their product?



Consider the following:

A software development company produces and distributes a product that automates record keeping in doctors' surgeries. As part of each surgery's support contract, the software company provides an annual upgrade of the product. The software support team maintains a database of all their support calls. This database is used as the basis for identifying modifications to be included in each upgrade. The following table lists the ten most common support issues.

Issue	Frequency	Issue	Frequency
Need to link members of the same family.	65	Cancelled appointments are not made available immediately.	29
No report available to list frequency of drug prescription by doctor (new government legislation).	59	Need to minimise application so other programs can be used without exiting.	24
Standard HP 6L printer driver not compatible.	58	Individual patient records can only be viewed on one terminal at a time.	18
Scheduling of doctors does not take account of multiple surgery locations within the one practice.	38	Program crashes when two referrals are printed for one patient.	17
Unable to import drug references from new Department of Health Version 7.1 database.	31	Loss of network connection is not notified until a save operation is initiated.	15



GROUP TASK Discussion
Categorise each of the above ten issues as either bugs in the application, hardware issues, other software issues or changes to program requirements. Explain why you have placed each issue into a particular category.



GROUP TASK Discussion
The above ten issues need to be prioritised. Should the frequency or number of times the issue has arisen be the sole criterion for deciding on its inclusion in the upgrade? What other criteria should be used? Decide on an appropriate order of priority for these ten items.

LOCATION OF SECTION TO BE ALTERED

Once a decision has been made to modify a particular aspect of a product, the location of the section to be altered needs to be determined. Models of the system are used to assist in this process. Structure charts and data flow diagrams will assist in the location of the module or modules requiring modification. Once a module has been identified, the programmer can analyse the original algorithms and IPO diagrams, and the actual source code. A thorough understanding of the original source code is required before any modifications are undertaken.



Consider the following:

A decision has been made to modify a customised book reseller program. One of the modifications required involves automatically reordering a predetermined number of a specific title when the stock held is insufficient to fulfil an order. Currently only the required number of books is ordered if insufficient stock is held in the warehouse. The programmer employed to complete this task is currently analysing the original data flow diagrams used to model the system. *Fig 7.1* shows the level one data flow diagram produced by the original developers of the product.

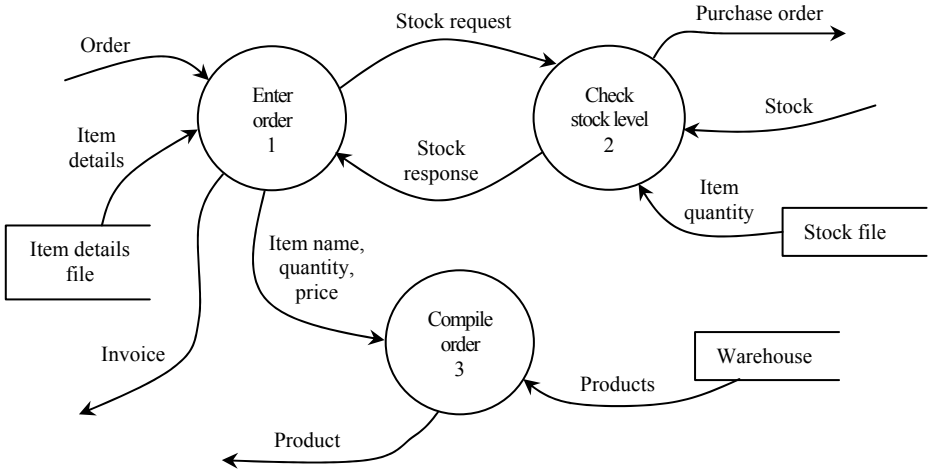


Fig 7.1
Level 1 data flow diagram for the book reseller problem.

The programmer determines that the relevant code is held in the Check Stock Level module. The data flow diagram for this module is reproduced in *Fig 7.2*.

After analysing this data flow diagram, the programmer decides to investigate module 2.2 dealing with insufficient stock. This module seems to generate the data for the purchase order to the supplier. This seems the most likely module in which to implement the desired modification.

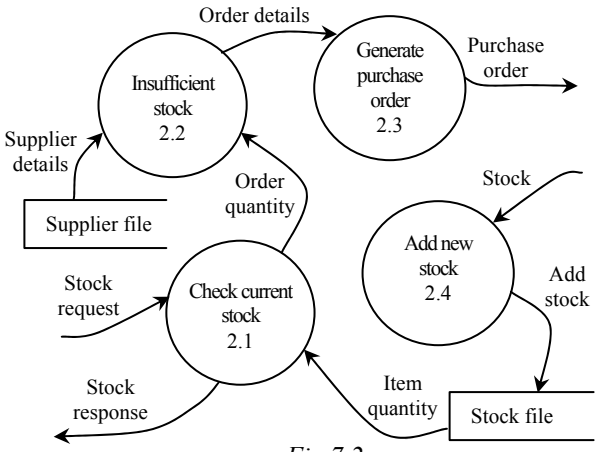


Fig 7.2
Level 2 data flow diagram for the Check Stock Level module.



GROUP TASK Discussion
Why do you think the programmer has selected module 2.2 for possible modification?

An IPO chart exists for each module of this program. The programmer examines the IPO chart for module 2.2. This IPO chart is reproduced below in *Fig 7.3*.

IPO CHART

Client: Quality Literature Pty. Ltd.
Project: Book Reseller
Module: Insufficient Stock 2.2

Date: 15/4/2001
Programmer: Melissa Davis
Page: 6 of 18

Input	Process	Output
Book code, Order quantity	Open supplier file for reading	
	Check book code exists in supplier file	
	Check book is available	
	Locate supplier associated with Book Code	
Supplier Details (from Supplier file)	Retrieve supplier contact details	
	Combine required order details: Supplier address, book name and code, quantity	Order details

Fig 7.3
IPO Chart for the Insufficient Stock 2.2 module.

The IPO chart describes the inputs, processes and outputs that take place in the Insufficient Funds 2.2 module. The programmer can now easily find the source code for this process. In the next section, we examine the actual changes made to the source code.

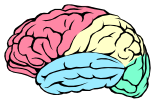


GROUP TASK Discussion
Examine the IPO chart shown in *Fig 7.3*. Using this chart, describe the general nature of the data held in the Supplier File.

DETERMINING CHANGES TO BE MADE

After the section of code to be modified has been located, we need to determine the changes to be made. Depending on the nature of the modification, changes may be required to data structures, files, and the user interface, as well as to the source code.

The consequences of changes made to one module must be considered. If a record data structure is altered then what effect will this have on other modules that access this data? For example, if a field within a record that once contained Boolean data is changed to hold integer data then there will be consequences for all modules that access these records. We don't want to solve one problem only to cause a number of new ones. Analysis of the original documentation should alert programmers to the possible consequences of changes.



Consider the following:

Let us return to our customised book reseller program. The modification involves reordering a predetermined number of books when stock is insufficient to fulfil an order. The module has already been identified for modification.

The programmer determines that changes need to be made to the Supplier file. This file contains a list of books distributed by each supplier. The modification requires the addition of a new field to this file. The quantity of each book to reorder needs to be stored. For example, consider the case of a particular book being ordered in quantities of 50 copies: if a new order is placed for 30 copies and only 10 are in stock then 50 will be ordered from the supplier. The existing Supplier file is read into an array of records with the following defined type:

```
TYPE
   BookCodeType=ARRAY[1..12] OF Char;
   SupplierRecType=RECORD
     Company   : ARRAY[1..40] OF Char;
     Address1  : ARRAY[1..40] OF Char;
     Address2  : ARRAY[1..40] OF Char;
     Suburb    : ARRAY[1..30] OF Char;
     Postcode  : ARRAY[1..4] OF Char;
     Phone     : ARRAY[1..40] OF Char;
     Fax       : ARRAY[1..40] OF Char;
     Email     : ARRAY[1..30] OF Char;
     Bookcode  : ARRAY[1..200] OF BookCodeType
   END;
   SupplierType=ARRAY[1..500] OF SupplierRecType;
```



GROUP TASK Discussion

Make up a series of ten records of type SupplierType. How are the books sold by each supplier, recorded using this record type? How do you think details of individual books are stored in this program? Discuss.

At present the BookCode field is used to store the ISBN number of each unique title sold by a supplier. The programmer decides to change the BookCodeType to a record containing the existing BookCode field together with the new field. The following type declaration accomplishes this task.

```
BookCodeType=RECORD
    ISBN        : ARRAY[1..12] OF Char;
    Reorder     : Integer;
END;
```



GROUP TASK Discussion
Modify the ten records produced in the last Group Task to include values for the new field. Do you think the programmer has made a wise choice by including the Reorder field in this file and data structure? Discuss possible alternatives.

After further analysis of the original documentation, the programmer finds the Suppliers file is accessed by one other module within the application: the data entry module, where new suppliers are entered and existing suppliers' details can be edited. This module and its user interface will require modification before the upgrade is released.

Next, the programmer finds and examines the algorithm written for the original Insufficient Stock module. They determine that changes to the quantity ordered can be made on the basis of the new fields added to the Suppliers file. The algorithm produced to accomplish this task is shown in *Fig 7.4*.

The problem becomes more complex than first thought. Imagine an order is placed for 55 books and the Reorder amount for this title is 20. If there are 9 books in stock then a request is sent to the Insufficient stock module for 55-9=46 books. We cannot merely order 20 books; in fact we need to order 60 books, 3 lots of 20. The algorithm in *Fig 7.4* deals with this situation.

```
Set NumToOrder to Reorder
WHILE OrderQuantity > NumToOrder
    Add Reorder to NumToOrder
ENDWHILE
Set OrderQuantity to NumToOrder
```

Fig 7.4
Algorithm for modifications to the Insufficient stock module.



GROUP TASK Discussion
Examine the algorithm in *Fig 7.4*. Perform a desk check with various values for Reorder and OrderQuantity. Explain in words how this algorithm accomplishes its required task.



GROUP TASK Discussion
The last line in the modification sets the OrderQuantity equal to the value held in NumToOrder. Suggest reasons why this line exists. Discuss other methods of achieving the same result.

IMPLEMENTING AND TESTING THE SOLUTION

Once the changes to be made have been determined, these changes must be implemented within the existing application. In many cases, changes will be made to the source code and the application will be recompiled, tested and distributed. In other cases, a small application or patch may be written to implement the changes on end-users' systems. Custom systems, written using COTS products, can often be changed directly on the site or over an internet link. This is particularly the case with script and macro modifications. In many cases, scripts and macros are not compiled until runtime, so the source code is available for change at the end-user's site.

The implementation of modifications can have a ripple effect to other aspects of the application. These problems are minimised if the original product was designed using a structured modular approach. When each procedure or function has been designed independently then the effect of changes will be easier to identify and correct. Modules that are used in a number of places throughout the application should be modified in such a way that they retain their original processing, including input and output parameters. Changes to modules that alter parameters will require modifications to each higher-level calling module.

Testing of modifications should be performed using the same techniques employed for the development of new products. Often CASE tool scripts and test data used during the original testing can be reused. The tests should not be restricted to the modified code segments, but applied to all aspects of the application that are in any way affected by the change.



Consider the following:

A product written using Microsoft Access requires modification. This product is used to produce reports for a school. Using the existing product, levels of achievement for outcomes are input as either A, B, C, D, E or NA, which are displayed on the reports as Excellent, High, Substantial, Satisfactory, Elementary, or Not Assessed. The product is being modified to allow all input to be carried out using the numeric keypad. As a result the inputs will become 0, 1, 2, 3, 4 or 5.

Two modifications are required to accomplish this task.

1. The data type of the grade field needs to be changed to a number. In this case a single byte integer is of sufficient size. Appropriate data validation changes are also required.

   Fig 7.5 shows the original data dictionary for the ReportOutcomes table.

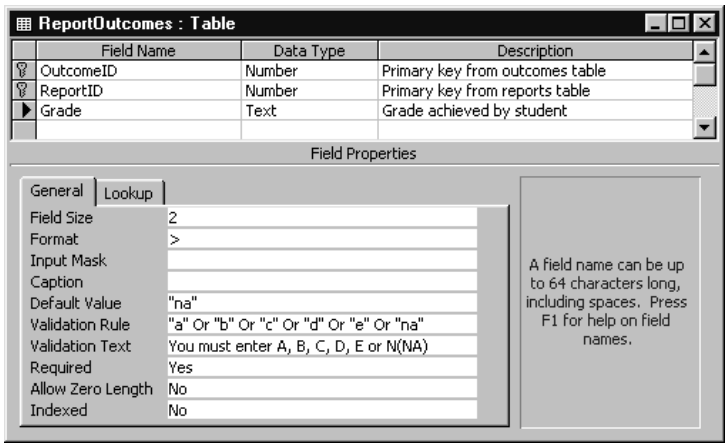


Fig 7.5
The original data dictionary for the ReportOutcomes table. Settings for the Grade field are shown.



GROUP TASK Discussion
Examine the data dictionary settings in *Fig 7.5*. List and explain all the changes that need to be made as a consequence of altering the Grade data type.

2. The function that converts the input into the required word needs altering.

   The programmer has located the section to be altered and has determined the required changes. The original function is shown in *Fig 7.6*. This function obtains the grade entered for each outcome and returns the appropriate word. The modification requires that the A, B, C, D, E and NA cases be replaced by 1, 2, 3, 4, 5 and 0 respectively.

```
Public Function OutcomeResult(Result)
  Select Case Result
    Case Is = "a", "A"
      OutcomeResult = "Excellent"
    Case Is = "b", "B"
      OutcomeResult = "High"
    Case Is = "c", "C"
      OutcomeResult = "Substantial"
    Case Is = "d", "D"
      OutcomeResult = "Satisfactory"
    Case Is = "e", "E"
      OutcomeResult = "Elementary"
    Case Is = "na", "NA", "Na", "nA"
      OutcomeResult = "Not Assessed"
    Case Else
      OutcomeResult = ""
  End Select
End Function
```

Fig 7.6
Original function converting grades into words for final reports.



GROUP TASK Activity
Make the changes to the code in *Fig 7.6* described above. Is it necessary to include the Case Else section in this function?



GROUP TASK Discussion
The programmer was considering allowing input to be either 0 or NA, 1 or A, 2 or B, 3 or C, 4 or D, or 5 or E. How could this have been achieved? What changes would have been required in the data dictionary? Do you think this would have been a better solution? Explain.

Finally, the modification requires testing. The user interface and the final reports must be thoroughly tested before the modifications are installed at the school.



GROUP TASK Discussion
Testing of this modification reveals a new problem. Statistics produced for the computer illiterate principal report the number of A's achieved by each student in the school. These statistics are produced by a query that was created by the school's computer teacher, who has since left to start his own software company! Is it the software company's responsibility to alter this query? Discuss from the points of view of the principal and the software company.

SET 7A

1. There are a number of ways of distributing and installing modified software. A piece of code inserted into an existing executable program to correct a problem is called a(n):
   (A) upgrade.
   (B) patch.
   (C) script.
   (D) bug fix.

2. Maintenance of software solutions is necessary to:
   (A) correct bugs in the source code.
   (B) modify the program to reflect changing requirements.
   (C) ensure software solutions remain compatible with other technologies.
   (D) All of the above.

3. In general, for large software products, maintenance programmers:
   (A) are usually members of the original development team.
   (B) control the configuration management of the system.
   (C) are rarely the same as the members of the original development team.
   (D) form part of the customer support team.

4. Modifications to particular procedures and functions should aim to:
   (A) maintain the original functionality of the code whilst introducing new features.
   (B) have few or no side effects on any routines that call the procedure or function.
   (C) allow existing data files to operate correctly once the modification is implemented.
   (D) All of the above.

5. Modifications to individual routines within software products:
   (A) can safely be tested in isolation from the remainder of the application.
   (B) require minimal testing because they are unable to affect other modules.
   (C) should only be made when there is sufficient demand for the modification from customers.
   (D) require testing in the same way as the original module.

6. Identifying the section of code to be altered is simplified when:
   (A) the original algorithms are available.
   (B) system models are accurately maintained.
   (C) the original test data is retained.
   (D) All of the above.

7. There are many reasons why software products require modification. Possible reasons include all of the following EXCEPT:
   (A) To correct bugs in the product.
   (B) Changes to user requirements.
   (C) Upgrades to operating systems.
   (D) To improve face-to-face training.

8. Modifying software products generally involves more than altering the source code. Other possible modifications include all of the following EXCEPT:
   (A) Alterations to documentation to reflect the changes.
   (B) Changes to data structures.
   (C) Examining competitors' products.
   (D) Creation or modification of testing and related test data.

9. The maintainability of a software solution can be best described as:
   (A) the usefulness of a software solution in regard to fulfilling users' requirements.
   (B) how easily a software solution can be maintained.
   (C) the speed at which modifications can be implemented into a software solution.
   (D) a measure of the ability of the product to evolve over time.

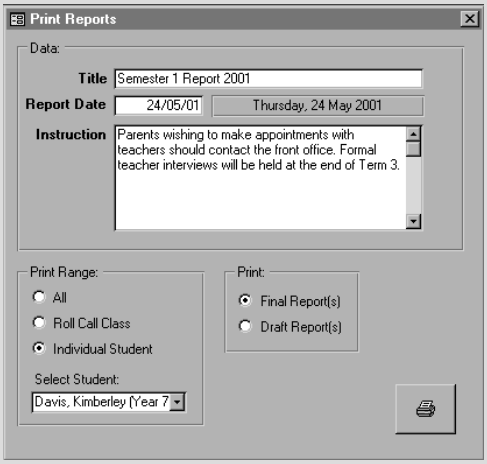
10. Prioritising corrections to be included in a product's upgrade requires consideration of which of the following?
    (A) Frequency of error: How many times have users encountered the error?
    (B) Impact of the error: Does the program crash? Is data lost or corrupted as a consequence of the error?
    (C) Cost and time to correct the error: Is it cost effective to correct? Can it be corrected within a reasonable time?
    (D) All of the above.

11. Maintenance of software products is said to be an evolving process. Software products continually change over time. Why is it necessary for software products to evolve over time? Discuss.

Use the following scenario to assist in answering questions 12 and 13.

A school's reporting software requires a number of modifications. Firstly, the school wishes to be able to print out the reports for a single year group. Secondly, there are now two different formats for the final report rather than the original, single format. Lastly, each report requires a range of dates indicating precisely the reporting period.

The screen shot at right shows the existing print dialogue for the school's reporting system. Maintenance programmers are currently working on creating the new report format, together with modifications so the range of dates is included on each format.



12. Examine the user interface above. Suggest alterations that should be made to this screen to implement the three changes. Justify and explain your reasons for each modification you suggest.

13. Some time later the school again contacts the software company for further modifications. They wish to be able to preview each report on the screen or print it. The school also has another three different formats they wish implemented. Explain how these modifications can be included on the print screen in such a way that future requests for new formats will not require changes to this screen.

Use the following scenario to answer questions 14 and 15.

A lawn and garden company designs, produces and distributes computerised watering systems. The software controlling the main control unit is currently being upgraded to take advantage of a new moisture sensor. The modification will cause the watering system to shut down when rain is detected. The moisture sensor returns a value every 30 seconds in the range 0 to 7 based on the volume of rainfall detected. Values of 2 or above indicate enough rain to shut down all sprinklers.



The system uses its own proprietary high-level language. The programmer has located the module requiring modification. This module has been reproduced at right. The value associated with the moisture sensor is returned and stored in Sensor(5). The solenoids used to turn the sprinklers on and off are controlled by setting the Boolean values contained in the array Sprinkler.

```
PROCESS UseSprinklers[Watering,TimeOn,TimeOff]
## Turn the sprinklers on
WHEN TimeOn > CurrentTime AND Watering=False
    Watering=True
    COUNT Index[1-10]
        Sprinkler[Index]=True
## Turn the sprinklers off
WHEN TimeOff > CurrentTime AND Watering=True
    Watering=False
    COUNT Index[1-10]
        Sprinkler[Index]=False
RETURN
```

14. Examine the UseSprinklers module, which is called every 30 seconds by the main program. Explain in words the processing occurring. Write down an example of a decision statement and an iteration statement in this language.

15. Modify the UseSprinklers procedure to work with the moisture sensor as described above. That is, for the water to be turned on the moisture sensor must have returned a value of 0 or 1. Similarly, the water will turn off, if it is on, if the moisture sensor returns values of 2 or higher.

DOCUMENTING CHANGES

Software products evolve over time, modifications and enhancements being added. These changes must be tracked and fully documented. Original documentation created during the development of the product must be modified to reflect any changes. This includes technical and user documentation. Large projects may contain many thousands of subroutines. Monitoring and tracking each of these subroutines and their modifications requires extensive documentation.



Configuration Management
Systematic control of changes to a software product, the purpose being to maintain software integrity, traceability and accountability during the life of the product.

Configuration management is the systematic control of changes to a product. For small projects developed by a small team of developers on a single site, configuration management may be a manual process. Often large projects will be developed and modified by many developers who are physically remote from each other, perhaps on the other side of the world or working for different companies. Configuration management CASE tools are available to automate the control and documentation of changes.

Version control is an important part of the configuration management process. Many development environments provide automatic version control. Each time a module is compiled the version number is incremented. More detailed version control is required for large systems maintained by a team of developers.



Consider the following:

A large software product has a client base of some 15,000 users and requires regular maintenance. The product is distributed in over 20 countries; each country has its own specific requirements in regard to functionality, particularly in terms of language. The software development company that maintains the source code has offices in 3 countries.



GROUP TASK Discussion
Maintaining a product with such a broad user base is a difficult logistical problem. Describe situations that make the maintenance of such a software product difficult.



GROUP TASK Discussion
Some modifications to this product are to be included in versions for all countries; other modifications are specific to individual countries. Discuss issues related to documentation that should be considered by those involved in the configuration management of this product.

SOURCE CODE DOCUMENTATION

Documentation of the source code includes comments and intrinsic documentation. In chapter 5 we discussed documentation in detail. The major function of source code documentation is to ease the job of maintenance programmers. Often documenting code at the time of writing is viewed as tedious, however at maintenance time this documentation will prove invaluable.

As source code is written the programmer obviously understands the logic. At the time they are written many statements within the source code will appear trivial however the code will not be so obvious when viewed by other programmers at a later date. Comments should be included that clearly describe what each code segment does. The code should be self-documenting so that it clearly explains how the code segment achieves its purpose.

Careful documentation of any changes made to the original code must be maintained. Details of who made the change, and when, should be included within the code. Any original comments that are no longer relevant should be removed or modified. It is vital that the integrity of the documentation be maintained throughout the modification of the product.



Consider the following:

A linear search function requires modification. At present the function finds only precise matches within the searched array. The modified function is to indicate a match if a part of an array element matches. For example, if the search item is 'at', then 'ate', 'mate' and 'at' should all result in a match.

The original function, coded in Visual Basic 6, is shown in *Fig 7.7*. The format of the output from the new function is to be identical to that of the original.

```
Public Function Linear(FindItem As String, InputArray() As String)
Dim Counter As Integer, TempOutput As String
'Initialise output to nothing
TempOutput = ""
'Linear search for the item
'Store found indexes as a comma delimited string
For Counter = 1 To UBound(InputArray)
   If InputArray(Counter) = FindItem Then
      TempOutput = TempOutput & Counter & ", "
   End If
Next
'Format output and set return value
If TempOutput = "" Then
   Linear = "Not found"
Else
   'Replace final comma with a period
   TempOutput = Left$(TempOutput, Len(TempOutput) - 2) & "."
   Linear = "Found at position " & TempOutput
End If
End Function
```

Fig 7.7
The original linear search function.



GROUP TASK Discussion
List and describe all the documentation features of the above function. Use examples from the function to explain each documentation feature.



GROUP TASK Discussion
Explain how the function could best be modified to implement the required change. What changes and additions to the source code documentation are required?

UPDATING ASSOCIATED HARDCOPY DOCUMENTATION AND ONLINE HELP

Printed documentation is becoming a rare occurrence as a means of distributing documentation. In most cases user and technical documentation is stored and distributed electronically. Electronic formats allow for more efficient editing and distribution of all types of documentation.

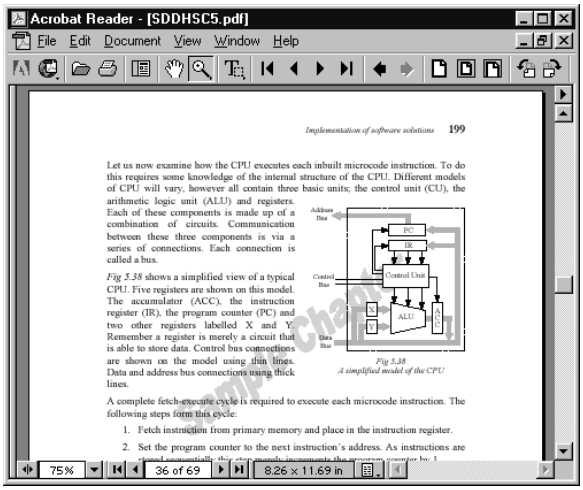
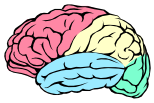


Fig 7.8
Adobe's Acrobat Reader is used to display and print PDF (Portable Document Format) files.

Adobe's Acrobat Reader is available free. This product allows portable document format (PDF) files to be read and printed. Often user manuals, as well as other traditionally printed documentation, are distributed using PDF or other similar formats. Prior to the widespread acceptance of these formats, hardcopy modifications were difficult and expensive to distribute. Often extra pages would be distributed that needed to be manually pasted into the existing user manual. Modified documentation can now be distributed in its entirety without the expense of printing and distribution.

Most software products now contain online help files. Most operating systems have their own standard format for online help files. They also contain an application to access these files. Most high-level languages now include development tools to assist in the integration of online help within products. Maintenance personnel must consider the effect on the integrity of online help of modifications made to products. Additions and modifications to the online help system should be made in order to reflect any modifications made to the application.



Consider the following:

The printer driver for a popular printer has just been upgraded to take advantage of features that are now available. Users can now choose print resolutions of between 600 and 1200 dpi in increments of 50 dpi. Previously either 600 or 1200 dpi had to be selected. The printer is also able to optimise the printout in three ways: text only, graphics only or mixed text and graphics. This option did not exist on the previous version of the driver.



GROUP TASK Discussion
The modifications to the printer driver must be reflected in the documentation. Describe documentation changes that should be made to ensure users are both informed of, and able to obtain support for, the new functionality.

USE OF CASE TOOLS TO MONITOR CHANGES AND VERSIONS

Maintaining accurate records of the maintenance process is a complex and time-consuming task. Computer aided software engineering (CASE) tools are available to automate this process. As we mentioned at the start of this section on documentation, the systematic control of changes to a software product is known as configuration management (CM) or software configuration management (SCM). Many CASE tools specifically designed to serve the process of configuration management are available.

Some of the functions commonly addressed in these CASE tools include:

- Change management

Change management supports the life cycle of each change, beginning with the initial change request to the inclusion of the modification into the final product. Each step of the software development process for each modification is monitored. For example, a history is maintained of each modification made, when it was made, and by whom.

- Parallel development

Maintenance of products by large teams of developers requires significant coordination. Parallel development CASE tools enable multiple users to work on the same project without fear of duplication or destroying another team member's work. Individual files are checked out as required by individual team members. This effectively locks the file, preventing others from making modifications. When the modification is complete, the file is checked in. The checking-in process releases the file and retains a record of the changes made. Distributed parallel development tools create scripts of changes made to each file; these scripts are distributed electronically to other team members. This ensures developers work with the most up-to-date code available.

- Version control

Version control CASE tools manage multiple versions of software components or modules. The system tracks which version of each component or module should be used when the application is finally reassembled. A record is maintained of prior versions of components, allowing the reconstruction of previous versions of the application.



Consider the following:

Merant's PVCS is a configuration management CASE tool. The screen dump at right shows a module being checked out for modification. The PVCS user interface looks and feels like a normal file management utility. Records are maintained of each modification made.

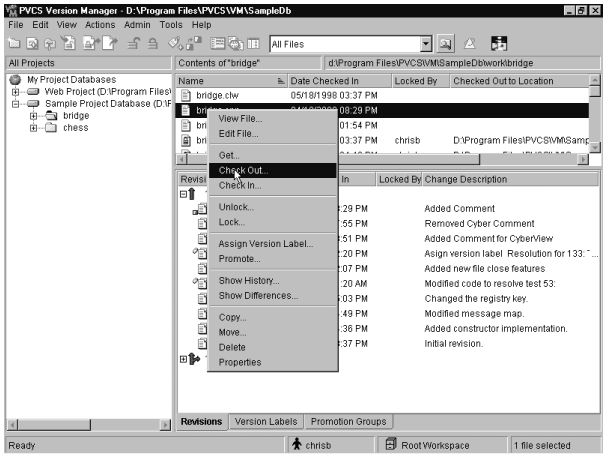


Fig 7.9
Merant's PVCS Configuration Management CASE tool.



GROUP TASK Investigation
Use the Internet to search for Configuration Management CASE tools. Make a list of the tools you find. For each tool write a brief description of its main features.



HSC style questions:

1. Which of the following best describes intrinsic documentation?

(A) Comments within the source code.

(B) System models, such as DFDs and structure charts.

(C) Meaningful variable, subroutine and function names.

(D) Data dictionary to describe all identifiers used in the source code.

Consider the following algorithm when answering Questions 2 and 3.

```
BEGIN IdentifyCustomer
    Get CustomerName
    IF CheckCustomer(CustomerName) THEN
        ProcessRequest
    ENDIF
END
```

2. Which statement must be true of the CheckCustomer subroutine?

(A) When called the subroutine alters the variable CustomerName.

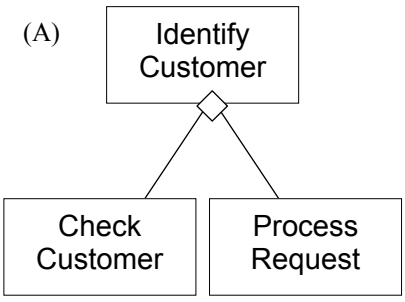
(B) It would be coded as a function that returns a Boolean value.

(C) The subroutine is only called when a legitimate customer name has been entered.

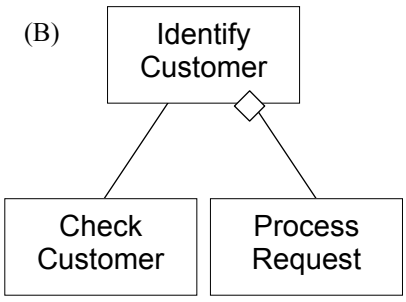
(D) It uses the customer name to determine the type of processing performed when the ProcessRequest subroutine executes.

3. The above algorithm was created from a structure chart. Which of the following best represents this structure chart?

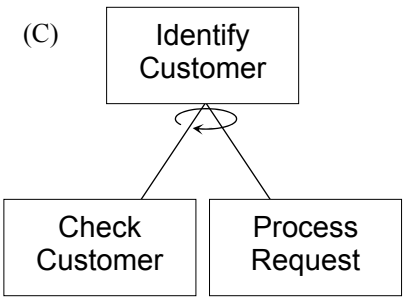
(A)



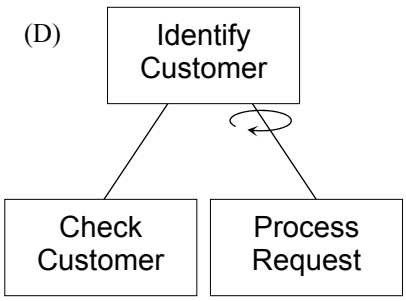
(B)



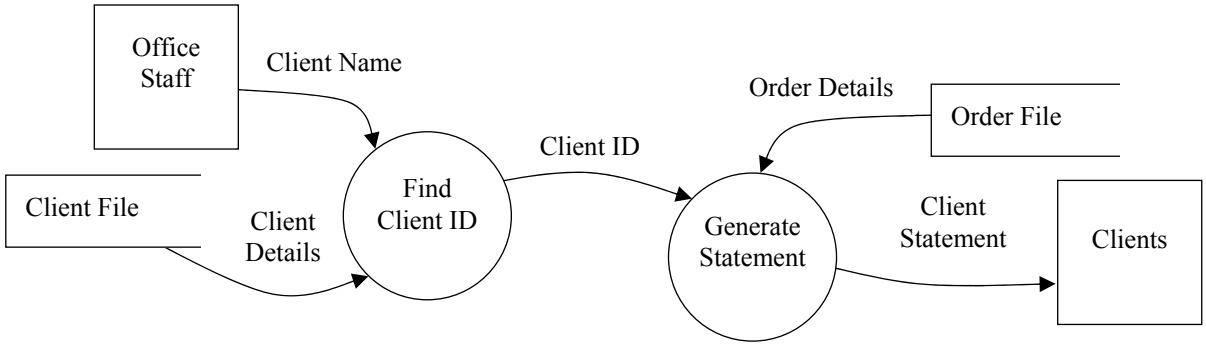
(C)



(D)



Refer to the following dataflow diagram when answering Questions 4 and 5.



4. When a client's name is misspelt by the office staff, a statement is inappropriately produced for the client with the highest Client ID. Where would the error in the system most likely be located?

(A) Office Staff entity

(B) Client File

(C) Find Client ID process

(D) Generate Statement process

5. Based on the above DFD, which of the following is the most UNLIKELY to be true?

(A) For each Client Name the Find Client ID process would read many clients' details.

(B) For each Client ID the Generate Statement process would read many orders.

(C) Each order within the Order File includes the client's Client ID.

(D) Each Client Statement includes the client's name.

Question 6.

Consider the following algorithm which will be coded as a function:

```
BEGIN doThis ( A(), D )
   B = A(0)
   FOR C = 1 TO D
      IF A(C) > B THEN
         B = A(C)
      ENDIF
   NEXT C
   RETURN B
END
```

(a) What is the purpose of this algorithm?

(b) Rewrite the algorithm using intrinsic identifiers.

(c) Name a standard sort algorithm where the above algorithm would be useful. Justify your response.

Question 7.

You have been contracted to make modifications to an existing software application that you did not write.

(a) Describe TWO examples of documentation that would be useful when making the modification. Justify your choices.

(b) Identify TWO examples of software debugging tools and explain how they would be used during the modification of the software application.

Suggested Solutions

1. (C) Intrinsic documentation is part of the actual programming code. Using meaningful identifiers is the primary form of intrinsic documentation. Other forms include indentation and also use of colour to help distinguish between language elements such as keywords, identifiers and operators.

2. (B) The call to CheckCustomer is also the condition for the decision, therefore it must return a Boolean value for the IF statement to operate correctly.

3. (B) The CheckCustomer call always occurs, whereas the ProcessRequest call is only made if CheckCustomer returns true. Note that within the structure chart the call line to CheckCustomer should contain the input parameter CustomerName and the returned Boolean value as a control parameter.

4. (C) It is likely that the Client File contains client records in ascending Client ID order as this is the order in which clients were added to the system. When a Client Name is not found the last Client ID to be considered would therefore be the highest Client ID.

5. (D) The Generate Statement routine receives the Client ID and retrieves Order Details for this Client ID, however it does not retrieve any data from the Client File which is where Client Names are stored. All other options are most likely true. For (A) the Find Client ID routine must search through many Client Details records. For (B) it is likely that individual clients will have many orders. For (C) Order Details must include the Client ID as it is the only data about the client which is known to the Generate Statement routine.

Question 6.

(a) The algorithm accepts two values, one being the array to be examined, the other being the number of elements in the array to be examined.

The algorithm then checks then examines each element in the array whilst recording the maximum value encountered. The function returns the maximum value found in the array.

(b)
```
BEGIN FindMax ( Array(), NumElements )
   Max = Array(0)
   FOR Index = 1 TO NumElements
      IF Array(Index) > Max THEN
         Max = Array(Index)
      ENDIF
   NEXT Index
   RETURN Max
END
```

(c) A selection sort is relevant, as it works by finding the largest element in the unsorted part of the array and swapping this with the current element (at the end of the unsorted part of the array).

This routine is useful as its logic will correctly find the largest element in the unsorted part of the array, and also the number of elements to be considered reduces by 1 with each pass as the number of elements in the sorted part of the array increases. This is taken into account well by decrementing the parameter representing the length of the array each time the routine is called.

Question 7.

(a) Internal documentation incorporated into the source code such as comments or intrinsic documentation such as meaningful variable and procedure names will make it easier for the programmer to understand exactly what the source code is doing. This makes it easier to determine which parts of the code need modifying to achieve the specified changes.

Systems documentation such as systems flowcharts, structure charts, data dictionaries provide an overview of the system and how the various parts interrelate. This provides the programmer with the information to determine the role played by individual components of the system, so they can better determine which parts need to be modified, and the effects of those modifications on other parts of the system.

(b) The software debugger would be used to determine what is happening as the program code executes, to help the programmer better understand what the code is achieving. In turn, this will help them to understand either how the original code works, or why their modifications do not work as expected.

The use of breakpoints allows the programmer to specify particular points in the program where the execution will temporarily stop, allowing them to view the contents of specified variables at that point, or to change the contents (such as a flag that has been incorrectly set by the code) so that execution can continue normally to test if subsequent modules work as expected.

Single line stepping allows the code to execute one line at a time so that the programmer can see the effect of individual lines of code on any variables and output produced. This may make it possible to see the display of required information that may otherwise have been immediately overwritten by some other command, for example.

CHAPTER 7 REVIEW

1. Maintenance of software solutions involves processes from which stage of the software development cycle?
   (A) Planning and design.
   (B) Implementation.
   (C) Testing and evaluation.
   (D) All of the above.

2. Configuration management can be best described as:
   (A) the systematic control of changes to a software product.
   (B) a series of CASE tools used to automate the maintenance process.
   (C) a way of ensuring changes to a software product are free of errors.
   (D) the techniques used to locate sections of code which require modification.

3. The major purpose of source code documentation is:
   (A) to provide a means for programmers to creatively influence the source code.
   (B) to enable future maintainers to more easily understand the source code.
   (C) to provide an audit trail. In this way the programmer responsible for a bug can be identified.
   (D) to allow users to better understand the processing taking place in the application.

4. CASE tools that allow multiple developers to work on a project simultaneously contain:
   (A) change management functionality.
   (B) parallel development functionality.
   (C) version control functionality.
   (D) recovery functionality.

5. Customised off-the-shelf (COTS) packages can greatly simplify and shorten the software development process. These COTS products can become a burden for future maintenance teams because:
   (A) new versions of the COTS package can affect the software in unexpected ways.
   (B) errors in the COTS package cannot generally be rectified by the software developer.
   (C) users with different versions of the COTS package may require different versions of the software.
   (D) All of the above.

6. Production of printed documentation is becoming less common. Which of the following reasons is the most important?
   (A) The cost of printing is too high.
   (B) Users rarely read printed manuals.
   (C) People prefer electronic forms of documentation.
   (D) Electronic forms are far easier to edit and distribute.

7. Maintaining the integrity of documentation is vital. Which of the following best describes the meaning of 'integrity' in this context?
   (A) The documentation should explain the reasons why modifications were made.
   (B) The accuracy of the documentation in relation to the product.
   (C) Documentation that considers all relevant social and ethical issues.
   (D) The documentation forms an integral part of the software package.

8. A series of processes is initiated by a series of keystrokes. The processes to be carried out are most likely stored as a:
   (A) macro.
   (B) query.
   (C) object file.
   (D) script.

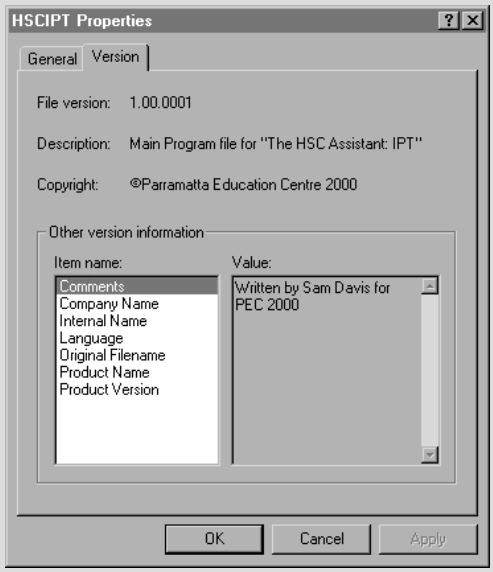
9. Locating the section of code to be modified usually involves:
   (A) examining system models and progressively isolating sections of these models until the required module or modules are identified.
   (B) searching the source code for keywords relevant to the process to be modified.
   (C) carefully examining user documentation.
   (D) using the product extensively to reproduce the circumstances of the error.

10. Modifying a software product involves the following step(s):
    (A) Identifying the reason for a change.
    (B) Locating the section to be altered and determining the changes to be made.
    (C) Implementing and testing the change.
    (D) All of the above.

11. Large software products usually have a full-time maintenance team assigned to continually implement modifications. Often these teams will receive many more modification requests than they can possibly perform. What criteria should be used to prioritise these requests? Suggest techniques and strategies that could be implemented to ensure the most urgent modifications are completed in a timely fashion.

Use the following information to answer questions 12 and 13.

Version information is usually stored as part of the executable files that form an application. This information is often displayed briefly as part of the application's initial splash screen at run time. Most operating systems allow you to view version information as part of a file's properties. For example, in MS-Windows, right clicking a file and selecting properties will display any version information.



12. Examine a number of executable files stored on either your own or one of the school's computers. Make a list of all the different types of version information commonly stored.

13. Most compilers provide options for setting version information along with copyright notices. This information then becomes part of the file's properties. Examine a compiler or development environment with which you are familiar. Describe how versioning is accomplished and the type of data that is stored.

The code at right has been reproduced from the raffle program written by Fred, as discussed in chapter 5. Use this code and the information that follows to answer questions 14 and 15.

The program requires modification to comply with the rules of a new raffle. In this new raffle there are three major prizes and ten consolation prizes. The strange thing about this raffle is that a ticket that wins one of the major prizes can also win a consolation prize.

14. Modify the Winners procedure to comply with the above requirements. (Also change the CheckWinner procedure so that only Year 12 students can possibly win one of the major prizes).

15. Fred wrote the original code. He seems to have the idea about intrinsic documentation, but comments within the code are certainly not his strength. Add appropriate comments to your modified code. Your comments in the CheckWinner procedure need not be completely honest!

```
Public PrizeWinners(22) As Integer
Public Sub Winners()
    Dim Winner As Integer, WinNum As Integer,
    CheckCounter As Integer
    Randomize
    WinNum = 1
    While WinNum <= 22
       Winner = Int(Rnd * 500) + 1
       PrizeWinners(WinNum) = Winner
       CheckWinner WinNum, Winner
       CheckCounter = 1
       While CheckCounter < WinNum
          If PrizeWinners(CheckCounter) = Winner Then
             WinNum = WinNum - 1
             CheckCounter = WinNum
          End If
          CheckCounter = CheckCounter + 1
       Wend
       WinNum = WinNum + 1
    Wend
End Sub

Public Sub CheckWinner(A As Integer, B As Integer)
   If Tickets(B).Year < 11 And Int(Rnd * 5) <> 0 Then
      A = A - 1
   ElseIf A < 3 And A + Tickets(B).Year <> 13 Then
      A = A - 1
   End If
End Sub
```

In this chapter you will learn to:

- define the problem and investigate alternative approaches to a software solution
- evaluate the ideas for practical implementation
- select an appropriate solution
- produce an initial Gantt chart
- use a logbook to document the progress of your project
- document the software solution
- generate a fully documented design for your project after communication with other potential users
- use and modify a Gantt chart as appropriate
- implement a fully tested and documented software solution in a methodical manner
- use project management techniques to ensure that the software solution is implemented in an appropriate time frame
- ensure that relevant ethical and social issues are addressed appropriately
- evaluate the project in relation to the original understanding of the problem
- review and evaluate the quality of the solution making necessary changes
- manage the project effectively
- communicate effectively with potential users

Which will make you more able to:

- explain the interrelationship between hardware and software
- differentiate between various methods used to construct software solutions
- describe how the major components of a computer system store and manipulate data
- identify and evaluate legal, social and ethical issues in a number of contexts
- construct software solutions that address legal, social and ethical issues
- identify needs to which software solutions are appropriate
- apply appropriate development methods to solve software problems
- apply a modular approach to implement well structured software solutions and evaluate their effectiveness
- apply project management techniques to maximise the productivity of the software development
- create and justify the need for the various types of documentation required for a software solution
- select and apply appropriate software to facilitate the design and development of software solutions
- assess the skills required in the software development cycle
- communicate the processes involved in a software solution to an inexperienced user
- use and describe a collaborative approach during the software development cycle
- develop and evaluate effective user interfaces, in consultation with appropriate people.

In this chapter you will learn about

Designing and developing a software solution to a complex problem

- defining and understanding the problem
  - identification of the problem
  - generation of ideas
  - communication with others involved in the proposed system
  - draft interface design
  - representing the system using diagrams
  - selection of appropriate data structures
  - applying project management techniques
  - consideration of all social and ethical issues
- planning and designing
  - algorithm design
  - refined systems modeling, such as:
    - IPO diagrams
    - context diagrams
    - data flow diagrams
    - storyboards
    - structure charts
    - system flowcharts
    - data dictionaries
  - additional resources
    - Gantt charts
    - logbooks
    - algorithms
    - prototypes
  - selecting software environment
  - identifying appropriate hardware
  - selecting appropriate data structures
  - defining files
    - purpose, contents and organisation
  - defining records
  - defining required validation processes
  - identifying relevant standard or common modules or subroutines
  - using software to document design
  - identifying appropriate test data
  - enabling and incorporating feedback from users at regular intervals
  - considering all social and ethical issues
  - communicating with others involved in the proposed system
  - applying project management techniques
- implementing
  - converting the solution into code
  - systematic removal of errors
  - refining the data dictionary
  - including standard or common modules or subroutines
  - using software to refine documentation
  - creating online help
  - reporting on the status of the system at regular intervals
  - applying project management techniques
- testing and evaluating
  - completing thorough program and system testing
  - completing all user documentation for the project
- maintaining
  - modifying the project to ensure:
    - an improved, more elegant solution
    - all needs have been met
    - the software solution operates under changed environments or requirements
  - updating the software specifications and documentation to reflect the changes

Whole project issues

- project management techniques
- social and ethical issues
- feedback from users at regular intervals

8

DEVELOPING A SOLUTION PACKAGE

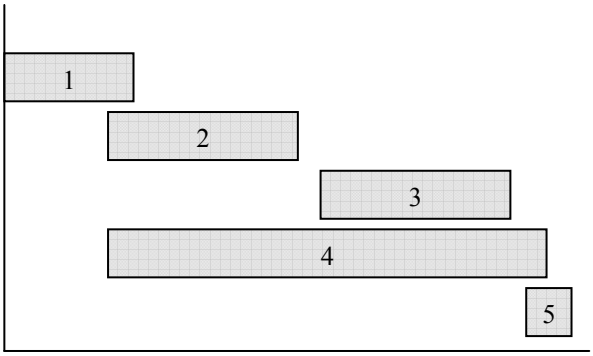
Completing projects is an important part of the Software Design and Development course. Project work aims to reinforce concepts and develop skills using a practical scenario. In most schools project work represents a significant portion of each student's assessment.

Working in teams is common practice in most software development environments. It is important that students are able to complete projects in conjunction with others. Advice from teachers and peers together with feedback from potential users will prove to be invaluable during the project's development.

Perhaps the most difficult part of many project assignments is deciding what it is you want to develop. In a commercial environment problems are usually presented to developers for solution, in this case you need to create or choose a problem to solve. Choose carefully, it is important you select a project that you will enjoy working on and that is achievable. If the final product of your labour proves useful or enjoyable for its intended audience then your efforts will seem more worthwhile.

In this chapter, we provide an example project. Be aware that projects focussing on different problems or utilising different tools may require the use of different techniques to those used in this project.

The software development cycle provides a framework for the development of the project. For our purposes we have split the development process into two distinct sections. Designing and developing the solution and implementing the solution. The first encompasses stages 1 and 2 of the software development cycle and the second stage 3 and 5. Stage 4 Testing and evaluation is an ongoing process and is therefore considered throughout. The Gantt chart in *Fig 8.1* describes the sequence of events. You should create an initial Gantt chart that includes specific dates for the completion of each stage of your project.



1. Defining and understanding the problem
2. Planning and designing
3. Implementation
4. Testing and evaluating
5. Maintaining

Fig 8.1
Gantt chart showing the relative timing for each stage of the software development cycle.



GROUP TASK Discussion
Create a Gantt chart to describe the timing of events for the development and eventual completion of your project. Discuss the appropriateness of your plans with the rest of your class.

DESIGNING AND DEVELOPING A SOFTWARE SOLUTION

LOGBOOKS

During the design and development process it is a good idea to maintain a logbook or process diary. This logbook describes each of the tasks and decisions completed together with notes in regard to issues encountered and solved throughout the development cycle. Often your teacher will require you to maintain a logbook as part of an assessment task as well as being an important evaluation and project management tool.

Logbooks can be maintained using a number of formats. Logbooks could include email messages, spreadsheets, database records, blogs, handwritten dated entries, electronic journal entries and a variety of other methods. The important feature of a logbook is the ability to create dated entries with ease.

Logbook entries for SDD projects could include the following details:

- date and time
- tasks achieved
- difficulties encountered and solutions
- ideas and thoughts
- reflection on progress
- upcoming tasks
- reference to resources used

DEFINING THE PROBLEM

After considering a number of possible ideas we settle on a project based on a ball bouncing around the screen. This seems like an achievable project that involves graphics and animation. The target audience is quite general, so we would like the final product to operate on MS-Windows machines and if possible be of a reasonable size so it can easily be distributed via the Internet.

Talking to friends and family generates a number of possible ideas that could be included in the product:

- The ball could change colour each time it hits a wall.
- Option to alter the size of the ball.
- Ability of the ball to move in different ways e.g. to simulate spin.
- Possibility to have more than one ball in action at a time.
- Need to have some competitive aspect to the product i.e. make it into a game.
- Don't have the ball in its own window; rather let it bounce around the entire screen i.e. it bounces when it hits any item on the screen.

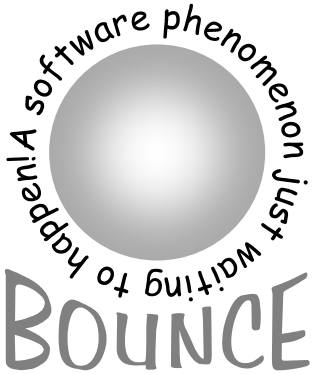


Fig 8.2
Logo for 'Bounce'

We decide the program would be a little boring if all it did was cause a ball to bounce around the screen. We wish to include an interactive and competitive aspect to the program. Eventually we come up with the idea that users could click on the ball. This would involve them chasing the ball around the screen with the mouse in an attempt to click it. If the ball is small and moves fast this would be difficult, however if the ball is large and slow then the task would be easy.

To achieve this requires options to alter the size and speed of the ball. We also decide to include an option to change the colour of the ball. We decide not to go ahead with the different movement idea or the multiple ball idea.

We like the idea of the ball not being in its own window, however this sounds like a rather difficult task. We'd have to know how to access the contents of an ever-changing screen. The problem will be simpler if we restrict the ball to its own window.

The following objectives are formulated:

1. The product is to run on MS-Windows machines using a standard Windows user interface.
2. The distribution files must be less than 1Mb in size.
3. The ball will bounce around within its own window.
4. Options exist to alter the size, speed and colour of the ball.
5. If the user successfully clicks on the ball then it stops and a message is displayed.
6. The ball starts bouncing immediately the game starts and continues bouncing until the user exits the program.
7. A 'New' option is included to return the ball to its default size, speed and colour.
8. A 'Help' option explains the aim of the game.

UNDERSTANDING THE PROBLEM

To understand the depth and complexity of our 'Bounce' project we commence creating models of the system.

Firstly, a storyboard is created on paper. This helps both in the design of the user interface and in understanding the complexity of the problem.

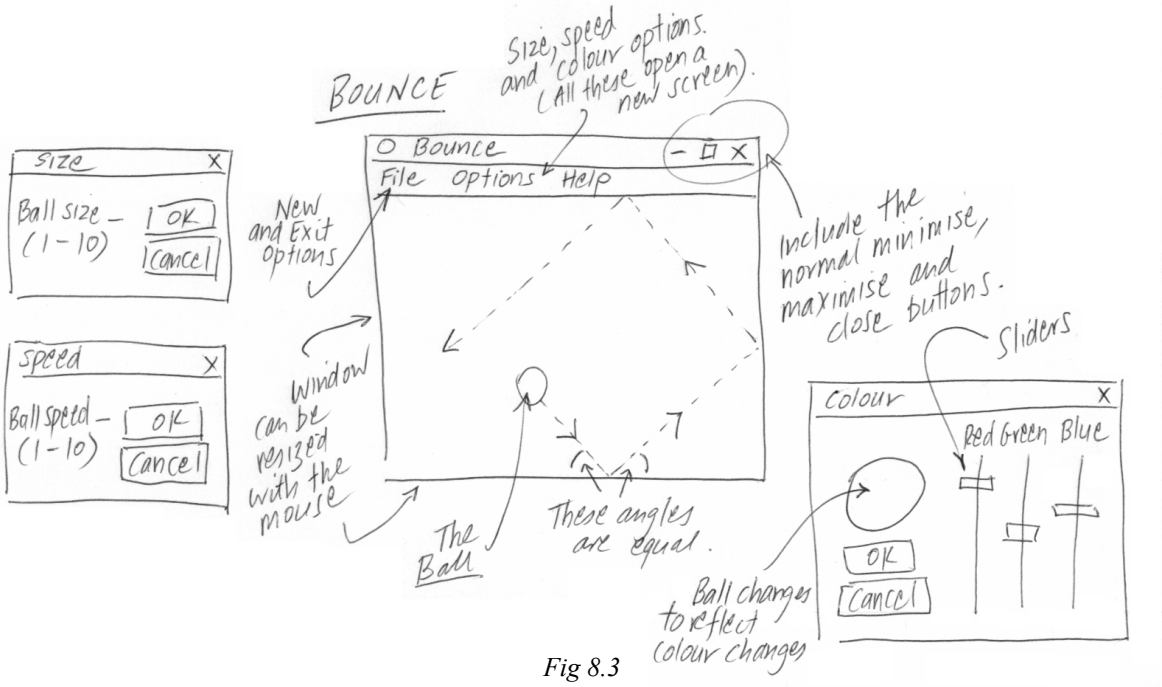


Fig 8.3
Storyboard showing the initial user interface design.

This user interface design is shown to a group of potential users (family and friends) for comment. As a result a number of modifications are made. The OK and Cancel buttons on the colour option screen are moved to the right hand side together with the simulated ball. Secondly, sliders are added to both the size and speed option screens. These changes improve consistency between all the option screens.

As 'Bounce' utilises various standard windows features we decide to use Visual Basic .NET. Visual Basic is a rapid application development environment that simplifies the development of event driven programs. Event driven programs respond to particular events that are usually initiated by the user e.g. opening a menu, clicking on a command button or, in our case clicking on the ball. The Visual Basic development environment allows us to 'draw' the user interface using a series of controls. Each control contains properties and events e.g. a command button might have its caption property set to equal 'OK', the button's click event would contain code to activate the appropriate subroutine and close the form.



GROUP TASK Discussion
Examine the storyboard in *Fig 8.3*. Brainstorm possible modifications that may enhance the "user friendliness" of this design.



GROUP TASK Activity
The storyboard does not detail the help screen design. Create a possible design for the help screen or screens.

To model the flow of data into and out of our program we use a context diagram. All inputs into our system come from the user. Apart from mouse clicks all user inputs will ultimately alter properties (or attributes) of the ball itself including its size, speed and colour. Mouse clicks from the user are processed and as a result the user receives a message if they successfully click on the ball. A possible context diagram is drawn (see *Fig 8.4*).

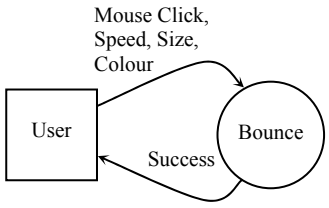


Fig 8.4
Context diagram for 'Bounce'.

Let us now try to understand the data requirements of our system. Because we are using Visual Basic much of the data will be held within the controls that form the user interface. For example, the properties within the ball include entries that describe its current position. We can access these properties to determine the ball's current coordinates and by altering these coordinates the ball will appear to move. At this stage, there appears to be no need to design any arrays or record structures. During the implementation stage, we will create a data dictionary to track each control property that directly affects our system's processing.



GROUP TASK Discussion
Explain the relationship between the context diagram above (*Fig 8.4*) and the storyboard (*Fig 8.3*). Describe in words the general nature of the processing required for the Bounce program.



GROUP TASK Discussion
Explain how the colour and size of the ball could be updated appropriately between the main bounce screen, the size screen and the colour screen.

PLANNING AND DESIGN OF A SOLUTION

In this section, we formalise our interface design, as we are using Visual Basic (a RAD product) we can create the actual user interface. We then create a series of algorithms, maintaining a data dictionary as we go.

During the creation of the user interface a number of further modifications to the original storyboard are developed. We add a simulated ball to the size option screen. This simulated ball allows the user to select a ball size visually rather than numerically. On the colour option screen we add the ability to alter the background colour as well as the ball colour.

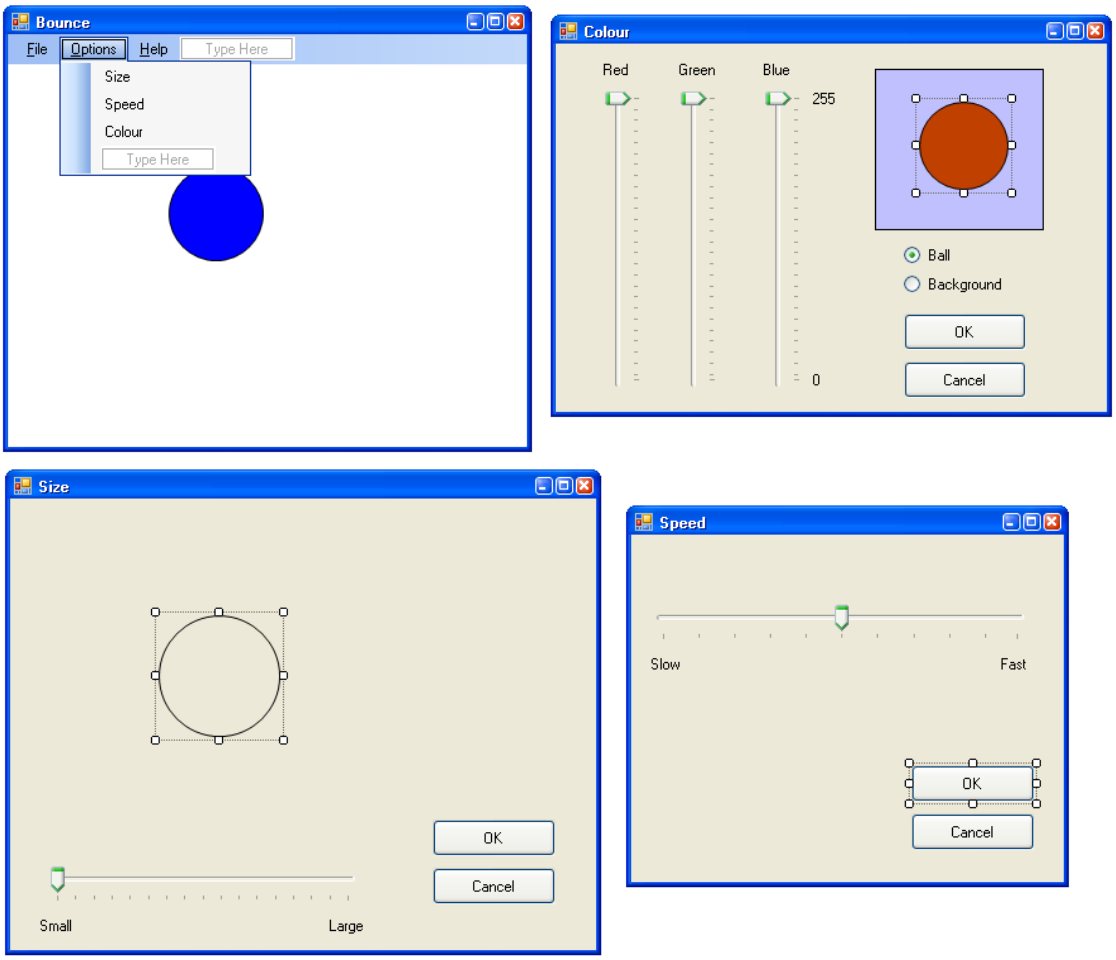


Fig 8.5
User interface design for 'Bounce'. Note changes and additions from the original storyboard.

The user interface is further developed into a limited prototype. This is accomplished by adding code to each of the menu items and OK buttons. For example, the Size menu option has the code `frmSize.ShowDialog()` entered as it's click event. This statement causes the 'Size' form to be displayed as a modal form. Modal means the user must close this form before accessing any other of the program's functions. The OK button's click events have the statement `Me.Close()` added. This closes the current form and removes it from memory. The prototype is used to obtain further feedback from potential users.



GROUP TASK Discussion
Describe the functionality of the limited prototype developed above. Do you think this prototype is a useful tool for obtaining user feedback? Discuss.

A data dictionary is now created using a word processor. Each control is listed and described next to the form on which it resides (see *Fig 8.6*).

On the colour form there are three trackbar controls (or sliders), one for each colour. Using 24 bit colour each red, green and blue component is represented using 8 bits. Therefore a range from 0 to 255 is possible for each of the three trackbars. The two radio (or option) buttons operate together so that only one can be checked at any time. Code will be required to synchronise the ball or background colour changes with the selected option button.

The ball, OK and Cancel buttons are permitted to have the same name on each of the forms. The scope of a control is determined by the form on which it resides.

This data dictionary will be updated should further data items be introduced during the development process.

Form	Control name	Description
frmBounce	shpBall	The ball
	mnuNew	File menu item
	mnuExit	File menu item
	mnuSize	Option menu item
	mnuSpeed	Option menu item
	mnuColour	Option menu item
frmColour	btnOK	Make change and close
	btnCancel	Close form
	shpBall	Simulated ball
	shpBackground	Simulated background
	optBall	Ball select
	optBackground	Background select
	tbrRed	Red trackbar (0-255)
	tbrBlue	Blue trackbar (0-255)
	tbrGreen	Green trackbar (0-255)
frmSpeed	btnOK	Make change and close
	btnCancel	Close form
	tbrSpeed	Speed trackbar
frmSize	btnOK	Make change and close
	btnCancel	Close form
	tbrSize	Size trackbar
	shpBall	Simulated ball

Fig 8.6
Data dictionary describing each control in 'Bounce'.



GROUP TASK Activity
Examine the user interface in *Fig 8.6* in conjunction with the data dictionary in *Fig 8.7*. Identify the placement on the user interface of each item in the data dictionary.

We now consider the creation of a set of algorithms. The most significant algorithm will be for the processing to actually move the ball. An IPO chart for this process is developed (see *Fig 8.7*).

We need to execute this subroutine repeatedly whenever the program is running. To do this, in an event driven environment, requires us to find an event that fires at regular time intervals. That is, no user intervention is needed for the ball to move. Visual Basic contains a timer control. This control type has a timer event that occurs after a particular amount of time has elapsed. We'll execute our MoveBall subroutine from one of these timer controls.

We could have put a loop around our MoveBall subroutine to cause it to be continually activated. However, this would not allow the program to process user inputs, as all its time would be spent moving the ball.

IPO CHART

Project: Bounce	Client: My Teacher
Module: MoveBall	Programmer: Me

Input	Output
Speed Size	Coordinates

Process

1. Get the ball's current position.
2. Check if the ball has reached one of the four sides of the window. (Considering the size of the ball).
3. Calculate the ball's new position.
4. Send the new coordinates to the ball.

Fig 8.7
IPO chart for the 'MoveBall' module.



GROUP TASK Discussion
Explain the sequence of events and processing that occur to animate the ball. Can you think of any other techniques that could have been used?

Let us now create an algorithm for the MoveBall module. To better understand the concept, we draw a diagram of the form containing the ball (see *Fig 8.9*). This diagram includes the dimensions influencing the ball's position within the form.

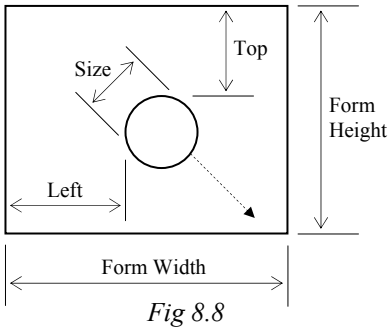


Fig 8.8
Form and ball dimensions.

The ball will change its X and Y coordinates by equal amounts. The value of the variable Speed contains this amount. If Speed is small then the ball moves slowly, if Speed is large then the ball will move faster.

We want the ball to change direction when it reaches one of the sides of the form. If the ball is moving down and right then both X and Y coordinates need to increase. If the ball then reaches the right hand side of the form then the X direction has to begin to decrease. If the ball reaches the bottom of the form then the Y coordinate has to begin to decrease.

We introduce a variable to store the direction the ball is travelling in both X and Y directions. If the ball reaches either the top or the bottom of the form then the Ydirection variable must be reversed. Similarly, if the ball reaches the left or right hand side the Xdirection variable must reverse.

It is possible that the ball could temporarily move outside the limits of the form. To solve this problem we need to change the X or Y coordinate to move the ball back to the edge of the form. This should occur as soon as we detect the ball is outside the bounds of the form.

Fig 8.9 shows a possible algorithm to perform these processes. Before continuing we add the variables used in this algorithm to our data dictionary. We also perform a desk check of the algorithm to ensure its correct operation.

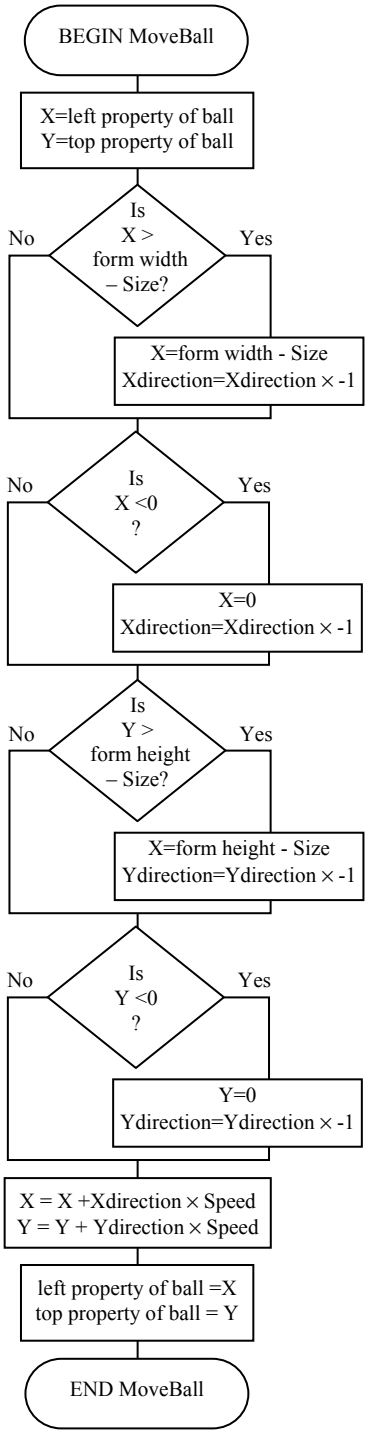


Fig 8.9
Algorithm for the 'MoveBall' module.



GROUP TASK Discussion
Create a series of appropriate test data items for the MoveBall algorithm. Desk check the algorithm using this test data.

Rather than create the remaining algorithms now, we decide to implement the MoveBall module in Visual Basic. We can return to the remaining modules once we have 'got the ball rolling' so to speak! Essentially, we are applying the planning and design, implementation and testing stages repeatedly for each module contained within the project.

IMPLEMENTATION OF A SOLUTION

In this section, we code our algorithm in Visual Basic. We have already created the user interface and developed it into a limited prototype. Our task is to add the functionality to complete the project. If the algorithms have been carefully created and tested then, in theory, the coding should be a relatively mechanical task.

Consider our algorithm for the MoveBall process. Coding this algorithm in Visual Basic results in the code shown in *Fig 8.10*. Note that this subroutine is within the class frmBounce hence Me refers to frmBounce and shpBall refers to the ball on the bounce form.

Remarks have been used throughout the code to explain each step. A general comment at the start of the procedure explains the overall purpose and requirements.

There are three variables that are used in this procedure that have not been declared within the procedure; Speed, XDirection and YDirection. These three variables need to retain their values for the life of the bounce form. As a consequence, they cannot be declared within the routine, rather we declare them at the form level. X and Y, which are declared within this subroutine, are only in existence for the life of an individual MoveBall call.

Before running our program we set initial values for Speed, Xdirection and YDirection. We also draw a timer control on our form and enter MoveBall as its timer event. We run our project and the ball actually bounces! Hurray! Also, if we resize the window the ball recognises the new window size and bounces accordingly.

```
Private Sub MoveBall()
    'Bounce shpBall around frmBounce
    'Requires Speed, XDirection, YDirection at form level

    Dim X As Integer
    Dim Y As Integer

    'Get current ball position
    X = shpBall.Left
    Y = shpBall.Top

    'The ball has reached the RHS of screen
    'Move back to edge and reverse X direction
    If X > Me.Width - shpBall.Width Then
       X = Me.Width - shpBall.Width
       XDirection = XDirection * -1
    End If

    'The ball has reached the LHS of screen
    'Move back to edge and reverse X direction
    If X < 0 Then
       X = 0
       XDirection = XDirection * -1
    End If

    'The ball has reached the bottom of screen
    'Move back to edge and reverse y direction
    If Y > Me.Height - shpBall.Width Then
       Y = Me.Height - shpBall.Width
       YDirection = YDirection * -1
    End If

    'The ball has reached the top of screen
    'Move back to edge and reverse Y direction
    If Y < 0 Then
       Y = 0
       YDirection = YDirection * -1
    End If

    'Increment or decrement X and Y values
    X = X + (XDirection * Speed)
    Y = Y + (YDirection * Speed)

    'Actually move the ball
    shpBall.Left = X
    shpBall.Top = Y
End Sub
```

Fig 8.10
The 'MoveBall' procedure in VB.NET.



GROUP TASK Discussion
Use example variables from the MoveBall subroutine to illustrate the idea and significance of variable scope.



GROUP TASK Discussion
Currently the ball moves at an angle of 45 degrees. Why is this the case? How could the MoveBall procedure be altered so the ball moves at a different angle?

The news is not all good. The ball seems to disappear off the screen at the bottom and under the menu at the top. It also appears to move too far to the right. After a frustrating amount of research and experimentation, we discover the cause of the error. The width and height properties for forms return the overall width and height as shown in *Fig 8.11*. We wish the ball to bounce within the internal section of the window, that is, the section that does not include the title bar, menu bar or borders. The origin, from the ball's point of view should be the internal section of the window below the menu strip. We discover forms have a ClientWindow property whose width and height accurately reflect the internal section of the window. Therefore we change Me.Height to Me.ClientWindow.Height, Me.Width to Me.ClientWindow.Width. We also need to consider the height of the menu strip as the ball reaches the top of the window. This involves changing the relevant section of code so it checks if Y is less than the height of the menu strip and if so then setting Y to this height. We test our program with the modifications and our diagnosis is confirmed. The problem is fixed!

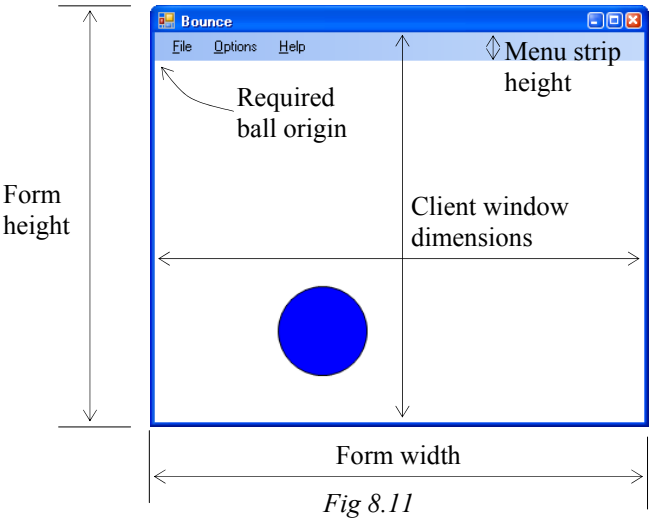


Fig 8.11
The form dimensions and the ball dimensions have different origins, resulting in an error.

It would be nice if we could get the section going that allows us to click on the ball. We decide to consider this section next.

We examine the IPO chart we developed (refer *Fig 8.12*). The original intention, described on our IPO chart, was that we'd read the current position of the mouse when every click event occurs. If this position is within the ball's current position and dimensions then we display a message to inform the user they've been successful. This would work; we could use the mouse down event for the Bounce form. This event returns the coordinates of the mouse. However, a simpler method comes to mind. Why not just use the click event for the

IPO DIAGRAM

Project: Bounce	Client: My Teacher
Module: Monitor Success	Programmer: Me

Input	Output
Mouse Click Coordinates, Size	Success

Process

1. Get coordinates of click position.
2. Get current ball coordinates and size.
3. Check if click coordinates are within the ball's current position.
4. If they are then display "Got me, ouch..."

Fig 8.12
IPO chart for the original 'Monitor Success' module.

ball itself? All we'd need to do is display a 'success' message every time this event occurs. A simple message box is used to display the message "Ouch, got me!" each time the ball is clicked.

Our code behind the form called frmBounce is reproduced below in *Fig 8.13*.

```
Public Class frmBounce

    Dim XDirection As Integer = 1
    Dim YDirection As Integer = 1
    Const Speed As Integer = 4

    Private Sub moveBall()
        'Bounce shpBall around frmBounce
        'Requires Speed, XDirection, YDirection at form level
        Dim X As Integer
        Dim Y As Integer
        'Get current ball position
        X = shpBall.Left
        Y = shpBall.Top
        'The ball has reached the RHS of screen
        'Move back to edge and reverse X direction
        If X > Me.ClientRectangle.Width - shpBall.Width Then
            X = Me.ClientRectangle.Width - shpBall.Width
            XDirection = XDirection * -1
        End If
        'The ball has reached the LHS of screen
        'Move back to edge and reverse X direction
        If X < 0 Then
            X = 0
            XDirection = XDirection * -1
        End If
        'The ball has reached the bottom of screen
        'Move back to edge and reverse y direction
        If Y > Me.ClientRectangle.Height - shpBall.Width Then
            Y = Me.ClientRectangle.Height - shpBall.Width
            YDirection = YDirection * -1
        End If
        'The ball has reached the top of screen
        'Move back to edge and reverse Y direction
        If Y < mnuStrip.Height Then
            Y = mnuStrip.Height
            YDirection = YDirection * -1
        End If
        'Increment or decrement X and Y values
        X = X + (XDirection * Speed)
        Y = Y + (YDirection * Speed)
        'Actually move the ball
        shpBall.Left = X
        shpBall.Top = Y
    End Sub

    Private Sub tmrSpeed_Tick() Handles tmrSpeed.Tick
        moveBall()
    End Sub

    Private Sub shpBall_Click() Handles shpBall.Click
        MsgBox("Ouch, Got me!", MsgBoxStyle.Information + MsgBoxStyle.ApplicationModal)
    End Sub
```

Fig 8.13
VB.NET source code behind frmBounce

```
    Private Sub ColourToolStripMenuItem_Click() Handles mnuColour.Click
        frmColour.ShowDialog()
    End Sub

    Private Sub SizeToolStripMenuItem_Click() Handles mnuSize.Click
        frmSize.ShowDialog()
    End Sub

    Private Sub SpeedToolStripMenuItem_Click() Handles mnuSpeed.Click
        frmSpeed.ShowDialog()
    End Sub

    Private Sub ExitToolStripMenuItem_Click() Handles mnuExit.Click
        End
    End Sub

End Class
```

Fig 8.13 (continued)
VB.NET source code behind frmBounce

We have examined in some detail the design and implementation of one of the modules within our project. The remaining modules still require algorithms and coding. Let us assume this has now been completed.

Our next task is to create the user documentation. As we intend distributing our product over the Internet, the user documentation needs to be in electronic form. We will include a help option within the program to describe t he operation and aim of the product. The website where the program can be downloaded will inform potential users of the general nature of the product together with system requirements.

Our final tasks are to compile the project into an executable file and then create a set-up program. The VB.NET environment provides a publishing wizard (*Fig 8.14*) that simplifies and automates the creation and deployment of the required setup files. Once the setup files are created, they require testing on a variety of Windows systems. Twenty family and friends perform this 'beta' test. The results are favourable and only minor changes are required. Comparing the final product to the original objectives confirms the product is a success. At last the product is complete! (Or is it?)

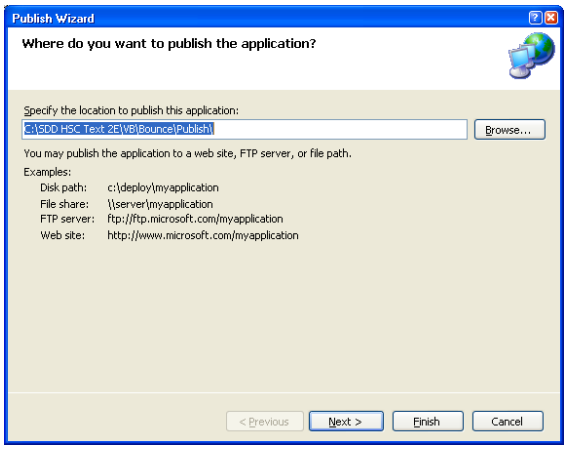


Fig 8.14
Screen from VB.NET publish wizard.

MAINTENANCE

Three months later, after the HSC exams are over (phew!), a few friends suggest modifications to improve 'Bounce'. They suggest including a timed play mode where the number of successful ball clicks during say a minute of play determines the player's score. The top ten scores could be maintained in a file. If a player achieves a top ten score then their name is input and both name and score are inserted appropriately within the top score file.



GROUP TASK Discussion
Discuss how the above timed play mode could be incorporated into "Bounce". Consider user interface additions/edits, how different speed and size balls will affect the game and an appropriate file format.



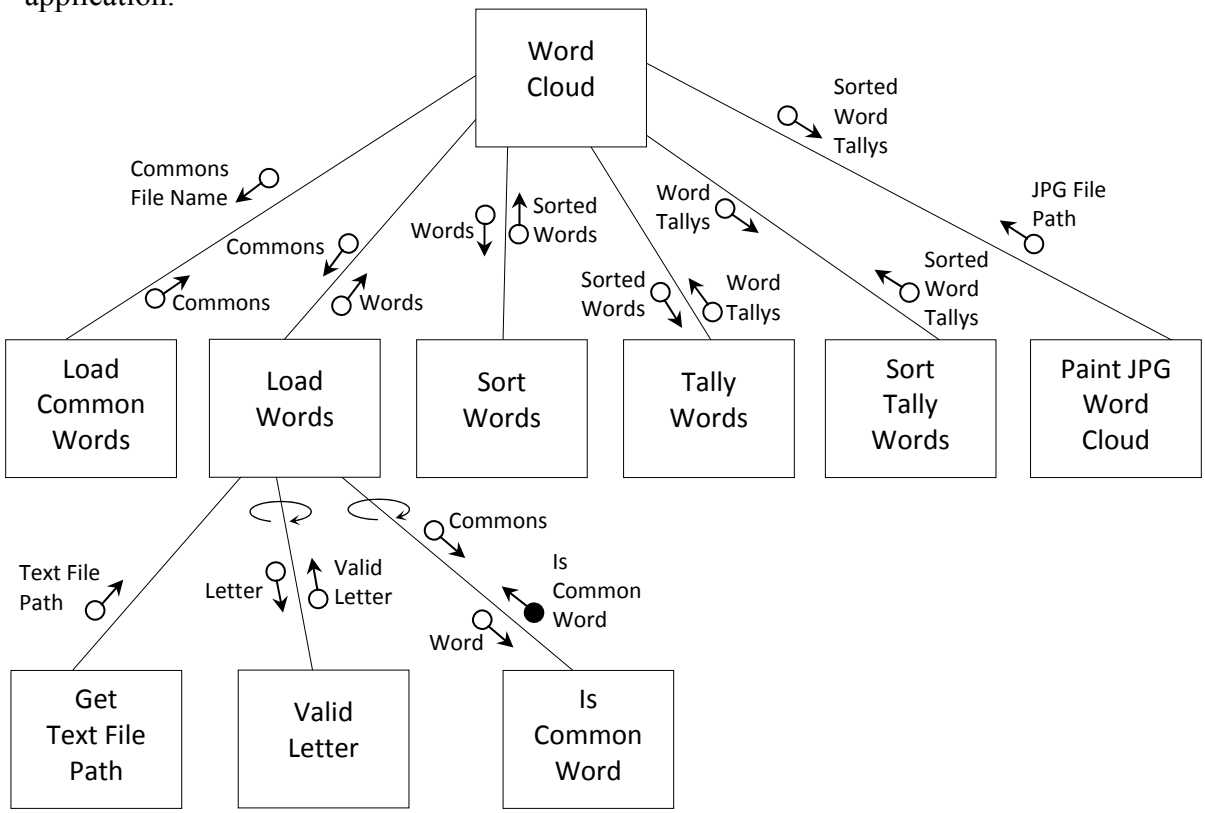
HSC style questions:

A WordCloud application is being designed and developed. A sample WordCloud based on the entire text of Shakespeare's play "Romeo and Juliet" is reproduced below.



The 50 most commonly occurring words are included in the WordCloud. The size and order of each word in a WordCloud represents the number of times the word occurs within the text. For example, "romeo" is the most commonly occurring word within Shakespeare's "Romeo and Juliet"; hence it is the first and largest word within the WordCloud. All words are converted to lower case. Common words (such as "and", "the", etc.) are excluded, and so too are words with less than three letters. All non-alphabetical characters are ignored.

The structure chart below describes the top-down design for the WordCloud application.



The following test data files have been prepared:

common.txt contains the words:

and
ate
can
food
the
then

sample.txt contains the text:

A CAT, BIRD AND A DOG.
The dog ate the cat's bird and then the dog's ant ate the dog's bird food.

Using the above files, the contents of data structures after each subroutine has been called will be as follows:

LoadCommonWords	
i	*Commons(i)*
0	and
1	ate
2	can
3	food
4	the
5	then

LoadWords	
i	*Words(i)*
0	cat
1	bird
2	dog
3	dog
4	cat
5	bird
6	dog
7	ant
8	dog
9	bird

SortWords	
i	*SortedWords(i)*
0	ant
1	bird
2	bird
3	bird
4	cat
5	cat
6	dog
7	dog
8	dog
9	dog

TallyWords		
i	*WordTallys(i)*	
	.Word	*.Tally*
0	ant	1
1	bird	3
2	cat	2
3	dog	4

SortTallyWords		
i	*SortedWordTallys(i)*	
	.Word	*.Tally*
0	dog	4
1	bird	3
2	cat	2
3	ant	1

PaintJPGWordCloud
wordcloud.JPG
dog bird cat ant

(a) Identify all inputs to and outputs from the WordCloud application.

(b) Add appropriate parameters to the following algorithm.

```
BEGIN WordCloud
   LoadCommonWords ( _____ )
   LoadWords ( _____ )
   SortWords ( _____ )
   TallyWords ( _____ )
   SortTallyWords (_____ )
   PaintJPGWordCloud ( _____ )
END WordCloud
```

(c) In reality the common.txt sequential file contains the 1000 most common English words sorted into ascending alphabetical order. Write an algorithm for the LoadCommonWords subroutine.

(d) An algorithm for the ValidLetter subroutine is reproduced below.

```
BEGIN SUBPROGRAM ValidLetter(Letter)
    ASCcode = ASCII code of Letter
    IF ASCcode >= 97 And ASCcode <= 122 THEN
        ValidChar = Letter
    ELSE
        IF ASCcode >= 65 And ASCcode <= 90 THEN
            ASCcode = ASCcode + 32
            ValidChar = Character represented by ASCcode
        ELSE
            ValidChar = ""
        ENDIF
    ENDIF
    RETURN ValidChar
END SUBPROGRAM
```

Outline, in words, the processing performed by the above algorithm.

(e) Propose and describe the operation of a suitable search algorithm for the IsCommonWord subroutine.

(f) Write an algorithm for the TallyWords subroutine.

(g) Given that:

- only the 50 most commonly occurring words are included in the WordCloud, and
- many text files will contain hundreds (or even thousands) of different words.

Propose and justify a suitable sort algorithm for the SortTallyWords subroutine.

Suggested Solutions

(a) Inputs: text file containing common words, text file containing the text to be analysed.

Output: the jpg image containing the word cloud.

(b)
```
BEGIN WordCloud
    LoadCommonWords(Commons File Name, Commons)
    LoadWords(Commons, Words)
    SortWords(Words, SortedWords)
    TallyWords(SortedWords, WordTallys)
    SortTallyWords(WordTallys, SortedWordTallys)
    PaintJPGWordCloud(SortedWordTallys, JPGFilePath)
END WordCloud
```

(c)
```
BEGIN LoadCommonWords(CommonsFileName, Commons)
   OPEN CommonsFileName for INPUT
   FOR i = 0 to 999
      INPUT Commons(i)
   ENDFOR
   CLOSE CommonsFileName
END LoadCommonWords
```

(d) The algorithm accepts a single letter and returns a value by assigning it to the variable ValidLetter. It converts the input letter to its ASCII code. If the ASCII code is in the range 97 – 122, which corresponds to lower case letters, then the letter is valid and is returned. If the ASCII code is in the range 65 – 90, corresponding to the upper case letters, then it adds 32 to the ASCII code, which converts it from upper case to lower case as each pair is 32 apart (for example A = 65, a = 97). It then converts the new ASCII code back to its corresponding letter which is returned. If the input character is not a lower case letter or upper case letter (i.e. not in the range 65 – 90 or 97 – 122), then a blank character is returned.

(e) As the Commons array is sorted in ascending alphabetical order, a binary search is the most appropriate search algorithm. It will require a maximum of 10 comparisons compared to an average of 500 on a linear search. The binary search will work by comparing the input Word with the middle (500^{th}) word of the Commons array. If the input Word is the 500^{th} element, it will return true. If the input word is less than the 500^{th} word (i.e. comes before it in the alphabet) then it will discard the top half of the array and inspect the middle of the bottom half of the array, i.e. the 250^{th} element. If the input Word is greater than the middle word in the Commons array, it will discard the bottom half of the array and check the middle of the top half, i.e the 750^{th} word. This process of halving the array continues until the word is found, or the range is narrowed to a single element, in which case the algorithm would return false.

(f)
```
BEGIN TallyWords(SortedWords,WordTallys)
   TallyIndex = 0
   Tally(TallyIndex).Word = Words(0)
   Tally(TallyIndex).Tally = 1
   FOR WordIndex = 1 To (Last index (Words))
      IF Words(WordIndex) = Tally(TallyIndex).Word THEN
         Tally(TallyIndex).Tally = Tally(TallyIndex).Tally + 1
      ELSE
         TallyIndex = TallyIndex + 1
         Tally(TallyIndex).Word = Words(WordIndex)
         Tally(TallyIndex).Tally = 1
      ENDIF
   ENDFOR
   WordTallys = Tally
END TallyWords
```

The above solution stores each word the first time it is encountered. Therefore initially (prior to the loop) the first word in the Words array (Words(0)) is stored and the tally for this record is initialised to 1. The loop is then able to compare each subsequent word (including the last word), if they match then the tally is incremented. If no match then we increment the TallyIndex, do the same as we did initially and continue iterating.

An alternative algorithm and strategy is as follows...

```
BEGIN TallyWords(SortedWords,WordTallys)
    CurrentIndex = 0
    CurrentWordTally = 1
    n = last used index of (SortedWords)
    FOR i = 0 to (n – 1)
        IF SortedWords(i) = SortedWords(i+1) THEN
            Increment CurrentWordTally
        ELSE
            WordTallys(CurrentIndex).Word = SortedWords(i)
            WordTallys(CurrentIndex).Tally = CurrentWordTally
            CurrentWordTally = 1
            Increment CurrentIndex
        ENDIF
    NEXT i
    IF SortedWords(n-1) = SortedWords(n) THEN
        WordTallys(CurrentIndex).Word = SortedWords(n)
        WordTallys(CurrentIndex).Tally = CurrentWordTally
    ELSE
        WordTallys(CurrentIndex + 1).Word = SortedWords(n)
        WordTallys(CurrentIndex + 1).Tally = 1
    ENDIF
END TallyWords
```

The above solution waits and stores the last encounter of each word in the Words array. It continues comparing adjacent Words. If the pair are the same then we increment the CurrentWordTally. If they are different then we store the first of the two words last compared in the final array together with the value of the CurrentWordTally. We then reinitialise CurrentWordTally, move to the next record and continue iterating. However, this strategy means we drop out of the loop without finalising processing (and critically storing) of the last word. Therefore, this must be done after the loop.

(g) *The following solution assesses the suitability of all three syllabus sort strategies. The question does not require students to assess all sort strategies.*

A *selection sort* is an appropriate sorting algorithm. A selection sort operates by finding the maximum element in the unsorted part of the list, and swapping it with the first element in the unsorted part of the list (as a descending sort is being used). After 50 passes of the selection sort algorithm, the 50 largest elements will be at the beginning of the SortedTallyWords array, and the sorting algorithm can stop.

A *bubble sort* is a less appropriate sorting algorithm. After each pass, the largest element is shifted to the start of the SortedTallyWords array. Only 50 passes of the sorting algorithm are required, however each pass will include many swaps which will prove to be unnecessary when only the first 50 items are required to be sorted. The bubble sort is therefore more likely to take longer than the selection sort on the same initial set of words.

An *insertion sort* is inappropriate as the entire sort would need to be completed (with possibly thousands of passes) before we could be sure that the 50 largest elements are at the start of the SortedTallyWords array.

CHAPTER 8 REVIEW

1. As part of your documentation, you create a word processor document that comprehensively details each data item within the software product. What is the common name for this documentation?
   (A) user manual.
   (B) quick reference guide.
   (C) data dictionary.
   (D) source code reference guide.

2. An IPO diagram is used to describe
   (A) the data items used in the program.
   (B) the variables and controls that will be processed.
   (C) the data entering a process, the nature of the processing performed and the resulting information leaving the process.
   (D) the output generated by a process.

3. A Gantt chart is a useful tool that
   (A) describes the timing and sequence of events that need to happen to complete a given project.
   (B) describes the nature and purpose of a project.
   (C) describes the desired outcomes of a project in terms of the stated objectives.
   (D) describes the different tools that will be used in a project.

4. Beta testing can best be described as
   (A) testing of the final solution by personnel within the software development company.
   (B) testing of the final solution by the end users of the released solution.
   (C) testing of the inputs and outputs by the software developers.
   (D) testing of the final solution by a limited number of users that are external to the software developers.

5. Rapid application development is best described as
   (A) an event-driven language.
   (B) the process of outsourcing aspects of a software development project.
   (C) an imperative language.
   (D) a process that makes use of software packages to quickly build applications.

6. Objectives are best described as
   (A) machine code produced as a result of compiling a program.
   (B) needs expressed in achievable realistic terms.
   (C) individual component parts of a program.
   (D) the end result.

7. A rough drawing or drawings that model the look and possible connections of a proposed application's user interface is called
   (A) a storyboard.
   (B) a data flow diagram.
   (C) a user interface.
   (D) an IPO chart.

8. The stages of the software development cycle are:
   i. Testing and evaluating.
   ii. Defining and understanding the problem.
   iii. Planning and design.
   iv. Implementing.
   v. Maintaining.
   Which order should these stages normally be undertaken?
   (A) ii, iii, iv, i, v.
   (B) iii, i, v, ii, iv.
   (C) v, i, ii, iv, iii.
   (D) ii, iii, iv, v, i.

9. A prototype can best be described as
   (A) a set of rules that govern the processes.
   (B) a final release of the software system.
   (C) an enactable model or mock-up of a software system that enables evaluation of features and basic functions.
   (D) a specific type of language. Prototypes utilise there own programming environment.

10. An algorithm can best be described as
    (A) a mathematical equation.
    (B) a series of unambiguous instructions or procedural steps.
    (C) a list of variables and control structures.
    (D) all of the above.

Answer questions 11 to 15 in relation to the development of your software project. You need to have decided on the general nature of your project before completing these questions.

For each item within each of the following questions create a 'plan of attack' you can follow during the completion of your project. This plan can be used as an outline for the creation of your final project. Your project need not address all the items listed below, however you should consider each during the development of your project.

11. Defining and understanding the problem
    - identification of the problem
    - generation of ideas
    - communication with others involved in the proposed system
    - draft interface design
    - communication with others involved in the proposed system
    - representing the system using diagrams
    - selection of appropriate data structures
    - applying project management techniques
    - consideration of all social and ethical issues

12. Planning and designing
    - algorithm design
    - refined systems modelling, such as:
      - IPO diagrams
      - context diagrams
      - data flow diagrams
      - storyboards
      - structure charts
      - system flowcharts
      - data dictionaries
    - additional resources
      - Gantt charts
      - logbooks
      - algorithms
      - prototypes
    - selecting software environment
    - identifying appropriate hardware
    - selecting appropriate data structures
    - defining files
      - purpose, contents and organisation
    - defining records
    - defining required validation processes
    - identifying relevant standard or common modules or subroutines
    - using software to document design
    - identifying appropriate test data
    - enabling and incorporating feedback from users at regular intervals
    - considering all social and ethical issues
    - communicating with others involved in the proposed system
    - applying project management techniques

13. Implementing
    - converting the solution into code
    - systematic removal of errors
    - refining the data dictionary
    - including standard or common modules or subroutines
    - using software to refine documentation
    - creating online help
    - reporting on the status of the system at regular intervals
    - applying project management techniques

14. Testing and evaluating
    - completing thorough program and system testing
    - completing all user documentation for the project

15. Whole project issues
    - project management techniques
    - social and ethical issues
    - feedback from users at regular intervals

In this chapter you will learn to:

- identify the needs that led to the development of different paradigms
- recognise the issues associated with using an imperative approach to solve some problems such as Artificial Intelligence (AI) and computer gaming
- recognise representative fragments of code written using the logic paradigm
- recognise the use of the logic paradigm concepts in code
- interpret a fragment of code written using the logic paradigm, and identify and correct logic errors
- modify fragments of code written using the logic paradigm to incorporate changed requirements
- code and test appropriate solutions in a language using the logic paradigm
- assess the appropriateness of a software solution written using the logic paradigm against a solution written using an imperative approach
- recognise representative fragments of code written using the object oriented paradigm
- recognise the use of the object oriented concepts in code
- interpret a fragment of code written using the object oriented paradigm, and identify and correct logic errors
- modify fragments of code written using the object oriented paradigm to incorporate changed requirements
- code and test appropriate solutions in a language using the object oriented paradigm
- assess the appropriateness of a software solution written using the object oriented paradigm against a solution written using the imperative approach
- describe the strengths of the imperative, logic and object oriented paradigms
- identify an appropriate paradigm relevant for a given situation
- evaluate the effectiveness of using a particular paradigm to solve a simple problem

Which will make you more able to:

- differentiate between various methods used to construct software solutions
- explain the implications of the development of different languages
- explain the interrelationship between emerging technologies and software development
- identify needs to which software solutions are appropriate
- apply appropriate development methods to solve software problems
- select and apply appropriate software to facilitate the design and development of software solutions

In this chapter you will learn about

Development of the different paradigms

- limitations of the imperative paradigm
  - difficulty with solving certain types of problems
  - the need to specify code for every individual process
  - difficulty of coding for variability
- emerging technologies
- simplifying the development and testing of some larger software projects
- strengths of different paradigms

Logic paradigm

- concepts
  - variables
  - rules
  - facts
  - heuristics
  - goals
  - inference engine
  - backward/forward chaining
- language syntax
  - variables
  - rules
  - facts
- appropriate use, such as:
  - pattern matching
  - AI
  - expert systems

Object oriented paradigm

- concepts
  - classes
  - objects
  - attributes
  - methods/operations
  - variables and control structures
  - abstraction
  - instantiation
  - inheritance
  - polymorphism
  - encapsulation
- language syntax
  - classes
  - objects
  - attributes
  - methods/operations
  - variables and control structures
- appropriate use, such as
  - computer games
  - web-based database applications

Issues with the selection of an appropriate paradigm

- nature of the problem
- available resources
- efficiency of solution once coded
- programmer productivity
  - learning curve (training required)
  - use of reusable modules
  - speed of code generation
  - approach to testing

9

OPTION 1
PROGRAMMING PARADIGMS

The evolution and history of programming languages is traditionally presented as a chronological list commencing with first generation and progressing through second, third, fourth and concluding with fifth generation languages. This technique presents programming languages in order of their emergence and increasing user friendliness. Although this is a reasonable technique for the study of the evolution of programming languages, it does not present the different paradigms used in a coherent manner. In this chapter we will focus on programming languages in terms of different paradigms.



Paradigm
A philosophical or theoretical framework. A different approach under which laws, theories and generalisations are produced.

A paradigm can be thought of as a philosophical or theoretical framework. This framework is then used as the basis of a new way of viewing problems. Theories, laws and generalisations are developed in support of the initial paradigm. Eventually, practical and workable techniques emerge for operating under the new paradigm. In this chapter, we consider and investigate different programming paradigms, why they came into existence and what problems they aim to solve. We concentrate on the major features of two paradigms – the logic and object oriented paradigms. We learn to write some simple source code using languages based on each paradigm as we assess the strengths (and weaknesses) of each. We also compare the logical, object oriented and imperative paradigms in terms of appropriate use and programmer's productivity.



Consider the following:

Many problems can be solved using a variety of different strategies. Some of these strategies and techniques will be useful for the solution of various other problems; some may be of little further use. Different paradigms provide a structure for solving problems from different angles. Lateral thinking involves generating new associations using seemingly unrelated areas, and using these associations in new and original ways. A paradigm is a description of the new association so it can be formalised and reused.



GROUP TASK Activity and Discussion
Everyone in the class is to calculate the sum of the first 1000 counting numbers e.g. 1 + 2 + 3 +… + 1000. Discuss all the different techniques used to accomplish this task. Is any one technique the best? What makes one technique better than another?

DEVELOPMENT OF THE DIFFERENT PARADIGMS

Until relatively recently, the architecture of computer hardware had driven the development of programming languages. Although many different computer architectures are currently being developed, by far the most common is still the traditional von Neumann architecture first developed to calculate trajectories for bombs and shells during World War II. The von Neumann architecture was implemented in the ENIAC (Electronic Numerical Integrator and Computer). Probably every computer you have seen or used is based on the von Neumann concept (also known as the stored program concept). Essentially, this architecture separates data and processing. Data is sent to the CPU for processing and the result is sent back to memory for storage. The CPU is a sequential device; instructions are processed one at a time in a predetermined sequence. The von Neumann computer has lead to the development of procedural or imperative languages. In this course, we have studied the imperative paradigm in detail, as it remains widely used and many of its methodologies are useful foundations to mastering other paradigms.

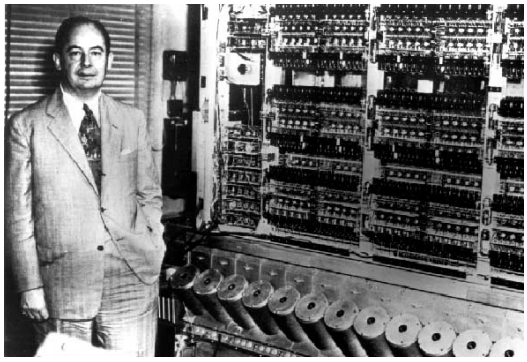


Fig 9.1
John von Neumann (1903-57) with ENIAC.

Imperative languages use sequencing, decisions and repetition as their main problem solving methods. Data and processing are separated with imperative languages. We create variables for data storage and then we perform processes on them. Each program has a beginning and a distinct end. This is not the only way of doing things.

Why would we want to do things differently? There are various motivations that have lead to the development of different paradigms. In the next section we consider some of these reasons.

LIMITATIONS OF THE IMPERATIVE PARADIGM

Although new imperative programming languages and user-friendly GUI interfaces have made the development of software an easier task, the resulting products are still performing the same tasks using essentially the same techniques. The difference is they are just doing it a lot faster! Imperative languages require the developer to understand all details of the problem and to be able to solve the problem completely. Many types of problems do not have precise solutions and developing an algorithm which solves such problems in not feasible.

The imperative or procedural paradigm on which languages such as Pascal, C, Fortran and Basic are based, restrict the developer in many ways. For instance, a function can only accept data as its inputs and it can only return data as its output. Individual programs are designed to solve a particular problem and they must be modified to solve related problems. The human brain can use the solution techniques used on past problems to assist in the resolution of new seemingly unrelated problems. As human beings we do not think in terms of data and processes; our brains are able to readily connect the two. The human brain is able to make inferences and deductions based on experience and intuition. Surely if we can create programming languages that can simulate the brain more accurately then productivity gains will result.

Difficulty solving certain types of problems

Computers were initially designed to solve mathematical and arithmetical problems. The finance and motivation for the development and production of early computers came from the military. This was soon followed by statistical analysis of census data and then by large business applications. These problems were best solved using mathematical techniques.

Many real life problems do not have definite answers or they have a variety of answers. Other problems can be solved using strategies that cannot be stated easily in strict mathematical terms. For example, doctors diagnose diseases in patients using symptoms and a certain amount of intuition gained from experience with other patients. Driving a car involves far more than knowing how to operate the controls. Good drivers are able to become an integral part of the vehicle; they sense potential dangers before they become problems. Communication between people involves multiple signals. Tone, volume and inflexions in voice have different subtle meanings; facial expressions and body language convey meaning. These types of problems are unable to be solved efficiently using imperative languages. A different set of rules governs the solution of these types of problem. A new programming paradigm that could help software developer's work towards solutions to these problems would be invaluable.



GROUP TASK Discussion

How would you explain precisely what a dog looks like to an alien? Make up a list of features that could be used to describe a dog. Would the alien then be able to differentiate a dog from a cat using your list? Can you program a computer to recognise a dog? In your experience are imperative languages suited to this problem?

The need to specify code for every individual process

The imperative paradigm requires that every part of the problem must be solved in complete detail before the final software will operate. For example, when using top-down design thorough testing cannot commence until the final lowest level subroutines have been written. Imagine the subroutine required to sort was not present within a database management system (DBMS). The whole application would fail even though this sort subroutine is a very minor part of the larger application.

Now imagine it was not necessary to specify details of every single process. Instead imagine the software could infer a suitable method of solution at runtime. If this were the case then software will be able to react to the changing needs of users without the need for modification. This is a significant aim of the logic paradigm and is why the logic paradigm is used to develop many artificial intelligence applications.

Difficulty of coding for variability

Many problems solved by software developers are similar in many ways or involve similar solution strategies. In chapter 4, we examined a number of standard algorithms for searching and sorting. There are countless other examples where programming tasks are repeated as part of the solution to multiple problems. Well-structured imperative programs can be designed so that modules of code can be readily reused; however this is not an integral part of the imperative paradigm. It would be preferable if code could be used to solve related problems without the need to be rewritten. This

reusability should be an integral part of the paradigm in much the same way as our brain uses past experience to solve new related problems.

The development of software products has now become one of the fastest growing industries. This was not the case a mere 20 to 30 years ago. It makes sense that software should be designed so that code can be reused in different contexts to solve different problems. Is there a paradigm that will encourage the reuse of code? If so, then the task of software developers will be simplified and the quality of the resulting products improved. Many other industries reuse components in a variety of contexts. For example, the padding in a lounge chair can also be used in the seat of a car or to provide sound- proofing in a studio. A 12-volt light bulb may be used in a torch, on a boat, in an airplane or in a car. Sheet aluminium is used to produce soft drink cans, lithographic printing plates and to line the outside of air transport crates. As software developers, we would benefit from programming languages that allowed us to reuse software components in a similar way.



GROUP TASK Discussion

Consider the problem solving methods you would use to plan a trip to a new location you have never visited before. Think about how you go about deciding on the best method of transport and then booking the trip so it will actually happen. Discuss aspects of your thoughts and actions that are known in every detail (imperative) and aspects which are not.

EMERGING TECHNOLOGIES

The first programmers had to program in machine language; a series of binary digits. The programmer was forced to think like the machine and the language was dependent on the processor. John von Neumann was 'using' his students to convert his code into binary machine language for input using punched cards. Apparently one of his students asked why the machine could not perform the conversion. Von Neumann replied that the resources of the computer where far too valuable to be wasted on such menial tasks. This must have made his students feel particularly valuable! Of course, eventually assembler languages emerged to perform this precise task. At the time, assembler languages were viewed as a way of removing the programmer from the technical aspects of implementation so they could focus on solving the problem. Compared to today's modern languages assembler is extremely primitive.

The ever-increasing speed of the technology underpinning computer hardware has allowed programming languages to further remove the programmer from the machine code. The emphasis is on creating programming tools to assist in the development of software products without the need for the programmer to directly interact with lower-level CPU processes. The final software product may not always be as efficient as a similarly developed machine code product, however its development is certainly a simpler and more intuitive process.

The imperative paradigm has evolved over the years from lower-level languages that directly accessed the computer's hardware functions. Computers are now powerful enough that different ways of viewing and solving problems can be utilised. In the next section, we look at some different approaches that are emerging to assist in the solution of problems.



GROUP TASK Investigation
The evolution of programming languages is often viewed in terms of generations; first, second, third, fourth and fifth. Using the Internet or other computer texts make up a table to explain the essential differences between each generation.



GROUP TASK Investigation
Hardware technologies have progressed at an astounding speed. Often the history of hardware is also viewed in terms of first, second, third, fourth and fifth generations. Make up a table outlining the major features of each of these hardware generations.



GROUP TASK Discussion
Examine your results from the above two investigations. Is there a correlation between the programming language generations and the hardware generations? Can you see how advances in hardware technologies have influenced software development? Discuss.

INTRODUCTION TO DIFFERENT PARADIGMS

The traditional imperative programming paradigm separates input, output, data, control and processing as distinct components. It is convenient to split computers into these components but is it the best or most appropriate way of doing things? The natural world, and in particular the human thought process, does not make this distinction. When we make a decision to get out of our chair and get a drink from the fridge we do use input, processing, output, data and control, however we do not separate them into distinct components rather we utilise all these components as a total integrated package. A fish is a complete system. Its component parts operate together transparently. The act of swimming combines input, processing, output, data and control into a network of actions culminating in the propulsion of the fish through the water. We need to consider new ways of combining and viewing components of computer systems. Perhaps there are better ways of integrating components that will assist software developers in the production of more 'natural' products.



Consider the following:

Suppose we wish to organise a dinner party for four friends. We need to know who is able to come so we phone each of our four friends. The final guest list is data, however the processes involved in obtaining this data are quite involved. For instance, perhaps you wish to ask Sue but if Sue is able to come then it's not appropriate to ask Bill. The process of asking Sue involves knowing her contact details, then contacting her. How do you contact her? By phone or by mail? If you choose mail then time constraints arise. Will there be time to ask Bill if Sue declines the invitation? You need all the components required to write a letter; paper, pen, envelope and stamp. You need to know what to write, how to get to the post box, some idea of the time it takes for a letter to be delivered, etc… We could continue on and on, however the point is that each process we perform is made up of a collection of building blocks that are combined to perform some task. Most tasks include building blocks that themselves are comprised of other building blocks.

Let us try to develop a list of building blocks used when we make decisions and solve problems such as the one outlined above. The building blocks for a particular task include differing types of components. They include knowledge or things we know. They include facts as well as how these facts are linked. They also include methods for manipulating our knowledge. In imperative computing terms, we call these processes, functions or procedures. We also have rules to follow. Often these rules are not stated but are understood e.g. it is not appropriate to ask Bill if Sue is coming.



GROUP TASK Activity
We have a set of components made up of facts, rules, processes and physical components. Make up a table with these building blocks as headings. Using the dinner party scenario complete the table by listing items under each of the building blocks.

There are various ways of approaching the solution to a problem. Each of these requires a set of components or building blocks upon which the approach or paradigm is based. The imperative paradigm we are all familiar with is based on variables and control structures. In this section, we introduce two other paradigms; logic and object oriented. These paradigms are built on different ways of solving problems using different components. To explain the concepts underpinning each of these paradigms let us consider a particular problem. Be aware that we are examining one particular problem and not all the paradigms are particularly suited to this problem's solution; nevertheless our example will serve as a worthwhile introduction.



Consider the following:

The Towers of Hanoi is a well-known problem, which was first published in 1883 by the French mathematician Edouard Lucas. The object of the game is to move all the disks to a different pole. You can only move one disk at a time and a bigger disk can never be placed on a smaller disk. There are three poles and initially the disks are all on one pole.



Fig 9.2
Initial set-up for the Towers of Hanoi problem using 8 disks.

The original problem allegedly contained 64 gold disks and it was thought that once all 64 disks had been moved, the universe would come to an end. The problem was to calculate an efficient manner of moving the disks to minimise the total number of moves. Once the total number of moves required is calculated then the life of the universe will be known.

Our aim in this section is not to come up with an answer but rather to illustrate different paradigms or approaches that could be used to arrive at a solution. Later in this chapter, we will examine how these paradigms are implemented using programming languages.

Imperative

If we were looking for an imperative method of solution we first need to consider variables. In our problem we would require variables to represent each pole, each disk and the pole on which each disk is currently sitting. We then require a series of control structures (sequences, decisions and repetitions) that could manipulate these

variables in such a way that the problem would be solved. Our choice of data structure used to represent each variable will have a large impact on the way we arrange our control structures into an algorithm, yet these two activities are separate.

We could create an array called Disk where for example Disk(3)=1 means the third smallest disk is currently on the first post. Another array called Post maintains the order of disks currently held on each post. Post(1,3)=5 might mean that disk 5 is currently on post 1 and is third from the bottom.

An algorithm is then created using control structures. The algorithm controls the movement of the disks to eventually solve the problem. When writing the algorithm we must have a precise understanding of how to solve the problem.

Logic

From a logic approach we focus on facts and rules. Some initial facts for this problem are that there are three poles, all the disks are on the first pole and each disk is a different size. The rules for this problem are that a larger disk can never be placed on a smaller disk, only one disk can be moved at a time and disks can only be moved from one pole to another.

Facts
- ⇨ There are three poles
- ⇨ Initially all disks are on the first pole
- ⇨ Each disk is a different size

Rules
- ⇨ Larger disks can't be placed on smaller disks
- ⇨ Only one disk can be moved at a time
- ⇨ Disks are moved from one pole to another

Fig 9.3
The logic paradigm is based on facts and rules. New unproven rules (Rules of thumb) develop from these facts and rules.

We could commence our exploration using the fact that all the disks are on the first pole, therefore our first move must involve moving the top disk. As a disk can only be moved from one pole to another, then this disk must be placed on the second or third pole. None of our rules help us to decide which pole so we take a guess and place it on the second pole. What move is next? Moving the second disk seems to make sense. It can't go on the second pole as it contains a smaller disk; one of the rules says this is not allowed. It can go on the third pole so we place it there. What options do we have now? Can we move the third disk? No. Can we move the smallest disk from the middle pole? Yes, it can move to either pole. Can we move the disk on the third pole? We could move it back to the first pole. Moving the smallest disk makes sense but to which pole? We consider each of these two possibilities. Eventually, moving the small disk to the first pole leads to a dead end so we continue down the other branch. We can learn from the dead ends and introduce 'rules of thumb' that will enable us to reduce the dead ends encountered.



GROUP TASK Discussion
The imperative and logic paradigms provide different approaches to problem solution. Discuss the essential concepts used in each of these paradigms.

Object oriented

Using an object oriented approach, involves considering each component of the problem as a self-contained unit. Each unit is able to remember things (data/attributes) and do things (methods/operations). There are two types of objects in this problem, disks and poles. We can approach the problem from the point of view of each of these objects.

If you were one of the disks then what do you need to know and what do you need to be able to do? Disks need to know how big they are, they need to know what pole they are on and if they are on top of the pole. They also need to ask poles if they can move to them and then know how to move to poles. The size of the disk, the pole it is on and whether it is on top is the data. Asking poles if they can move to them and moving to other poles are the methods.

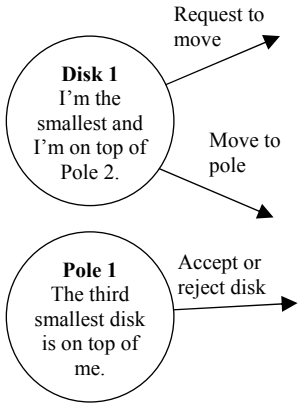


Fig 9.4
Diagrammatic view of the Disk and Post objects showing their data and methods.

Let us now look at a pole. Poles need to know what disks they have. In particular, they need to know the size of their top disk. This is their data. They also need to be able to reject disks trying to move to them that are larger than their top disk. This is a method.

The interaction of the disks and poles results in the eventual solution of the problem. Let us consider the interactions that may occur at the commencement of the solution using an object oriented approach. The top disk sends a message to the second pole asking if it can move there. The pole responds by checking its top disk. In this case, the pole tells the disk it can move. As a result the smallest disk moves to the second pole. The new top disk on the first pole realises it is now at the top so it attempts to move to the second pole. It is rejected. It then requests a move to the third pole and is accepted. The processing continues in this fashion until the final objective is achieved.

Our explanation above implies that methods are carried out in a sequential fashion. This is not the case. A number of disks can be simultaneously requesting moves. This is one of the aspects of the object oriented paradigm that makes it so powerful; each object is able to operate on its own. As soon as a disk reaches the top of a pole it can start asking other poles for permission to move. Each disk is in control of itself. Similarly, each pole may be receiving requests from a variety of disks. It is up to the pole to either accept or reject the requests.



GROUP TASK Discussion
CPUs designed using the von Neumann architecture process instructions in a strict sequence. How then can software written using object oriented programming languages be executed on these machines? Discuss.



GROUP TASK Research
The object oriented paradigm is the basis of many popular programming languages. Create a list of programming languages that are based on the object oriented paradigm.

SET 9A

1. A paradigm can best be described as
   (A) a new way of programming.
   (B) a philosophical or theoretical framework.
   (C) Darwin's theory of evolution.
   (D) a problem solving technique.

2. Which paradigm is based on variables and control structures?
   (A) Evolution
   (B) Logic
   (C) Imperative
   (D) Object oriented

3. Productivity can best be described as
   (A) a measure of the output produced from a given set of inputs.
   (B) the number of lines a programmer can write.
   (C) the collection and analysis of information.
   (D) a revised performance standard.

4. Which paradigm has evolved directly from lower level languages that directly accessed the computer's hardware functions?
   (A) Imperative
   (B) Object oriented
   (C) Logic
   (D) All of the above

5. Homer is writing a piece of code that is based on previously known facts and rules. Which paradigm would Homer most likely use?
   (A) Logic.
   (B) Functional.
   (C) Imperative.
   (D) Object oriented.

6. The imperative paradigm is also referred to as
   (A) the operational paradigm.
   (B) the repetitive paradigm.
   (C) the algorithmic paradigm.
   (D) the procedural paradigm.

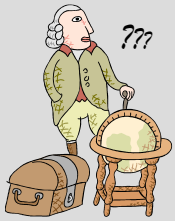
7. Which programming language is based on the object oriented paradigm?
   (A) Basic.
   (B) Fortran.
   (C) C++.
   (D) Pascal.

8. Roland owns a software development company that specialises in researching and creating family trees. Which paradigm would be most suited to specifying and determining relationships between people?
   (A) Functional.
   (B) Object oriented.
   (C) Logical.
   (D) Imperative.

9. What is another term commonly used for the von Neumann concept?
   (A) Stored program concept.
   (B) Data processing concept.
   (C) Sequential data concept.
   (D) Variable data concept.

10. Imperative languages use which components for solving problems?
    (A) data and methods.
    (B) facts and rules.
    (C) attributes and rules.
    (D) variables and control structures.

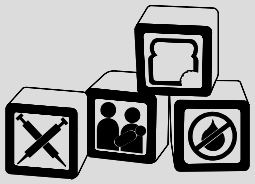
11. For thousands of years, man believed the earth was flat. We now know that this is not the case. In effect a paradigm shift affecting the entire population occurred. Explain the term 'paradigm shift'. Use the 'flat earth' scenario to assist with your explanation.



12. Most modern computers are based on the 'Stored program concept'. John von Neumann initially developed this concept. What is the stored program concept and how has it influenced the evolution of programming languages?



13. The imperative, logical and object oriented paradigms utilise different sets of components or building blocks. List and describe the building blocks used with each of these paradigms.



Use the following scenario to answer questions 14 and 15.

A man died leaving all his money to be divided amongst his widow, four daughters and three sons. He stipulated that each daughter should receive three times as much as each son and each son should receive twice as much as their mother. If the exact amount left was $7936, how much should the widow receive?



14. Which paradigm seems the most suited to the solution of this problem? Justify your choice by explaining why the other paradigms would be less suitable.

15. Using the paradigm you selected in question 14, attempt to solve the above problem. Concentrate on the process used to solve the problem rather than just the final solution.

PARADIGM SPECIFIC CONCEPTS

In this section, we examine the logic and object oriented paradigm in more detail. The important features of each paradigm are examined, together with examples using programming languages based on the paradigm. The aim being to illustrate how each paradigm is implemented in a programming environment.

Before we commence, a more detailed understanding of the term 'paradigm' should serve as a worthwhile introduction. Understanding new paradigms involves a shift in the way we think; often the term 'paradigm shift' is used to describe this change. A paradigm shift can be described as a global change in thinking. To achieve a paradigm shift involves coming to a new realisation through new insights. This is often not an easy task, particularly when old ways of thinking about a problem are thoroughly entrenched. We, as humans thought the world was flat, we now know this is not the case. We thought the earth was the centre of the universe; of course it is not. These are examples where the entire population of the world has undergone a paradigm shift.



Consider the following:

Charles Darwin did not release his book 'Origin of the Species' for many years after he had developed and refined his theory of evolution. He was concerned about the reaction he would receive from the general public. When Darwin eventually did go public, the majority of the population were not only slow to accept his theory but were outraged. Darwin's supporters were publicly humiliated and ostracised. Darwin's theory involved a paradigm shift, a new way of looking at creation.

Over time, a paradigm shift has occurred where evolution is now an accepted theory. As school students we are taught Darwin's theory of evolution almost as though it were fact; even the most sceptical of us accept it as a possibility.

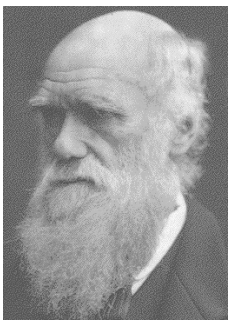


Fig 9.5
Charles Darwin's theory of evolution resulted in a paradigm shift.



GROUP TASK Discussion
Imagine if a new theory arose in regard to creation. How receptive do you think the population would be? Just because a paradigm is accepted this does not make it the only possibility nor is it necessarily the best. Discuss.

As software developers, we need to be able to choose a paradigm most suited to the current problem. Software development is primarily a problem solving activity; problem solving is a creative process requiring original thought and new realisations. Programmers need to equip themselves with a tool kit of ideas and strategies they can draw upon when embarking on the development of new products. Knowledge and understanding of programming languages that are based on alternative paradigms adds to this tool kit.

In the next section, you will need to have an open mind to the ideas presented. Try to view these new concepts in their own right yet balance their virtues with their short-comings. The aim is to enable you to select a programming language, or selection of languages, that provides the best tools to solve a particular problem. You will probably need to read much of what follows a number of times to grasp these concepts.

LOGIC PARADIGM

We saw in the previous section that the logic paradigm uses facts and rules as its basic building blocks. In this section, we extend our logic paradigm idea further to include concepts central to logical programming languages. The logic paradigm is used to develop programs in the area of artificial intelligence. Rather than just explain the concepts, we'll create a small program using Prolog and use this example to illustrate the concepts. Following this example, we examine expert system shells and how they can be used to create expert systems.

Prolog is perhaps the most common logical programming language hence examples throughout this section will use this language. Prolog is short for **PRO**gramming in **LOG**ic. Prolog was developed as a language in the early 1970s. One of the major influences on the nature of the Prolog language was the need to process natural language. Prolog is used primarily in the areas of artificial intelligence including natural language processing. There are various versions of Prolog in common usage. In this text, we will use 'Strawberry Prolog', a product developed by Dimiter Dimitrov Dobrev in Bulgaria. At the time of writing, a demonstration version was available for free download on the Internet. SWI-Prolog is another popular version which is freely available. For our purposes the particular implementation or version is not critical so we will just call it Prolog.

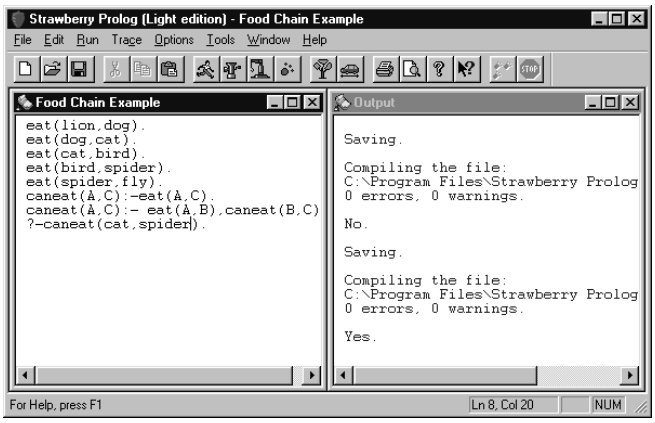


Fig 9.6
The Strawberry Prolog editor and user interface.



Consider the following:

Let us assume we have a simple food chain where lions eat dogs, dogs eat cats, cats eat birds, birds eat spiders and spiders eat flies. In Prolog we write the following facts:

```
eat(lion,dog).
eat(dog,cat).
eat(cat,bird).
eat(bird,spider).
eat(spider,fly).
```

In Prolog `eat` is called a predicate; to be more precise it is a user defined predicate. We just made up the name `eat`; we could have used `devour` or almost any other name. The animal's names are atoms. These are simple data items that are known in Prolog as atoms. Atoms must commence with a lower case character. A fact is a definite known item e.g. cats eat birds or in Prolog `eat(cat,bird).`



GROUP TASK Activity
Design a set of facts, using Prolog syntax, describing the components of a computer system as either input, processing or output components. Remember that some components are both input and output devices.

Querying these facts is simple. Say we wish to determine if a spider can eat a dog. We enter the line `?-eat(spider,dog).` When we run our program (or query), the Prolog inference engine responds with `No`, where as if we enter the query `?-eat(dog,cat)` the output is `Yes`. In Prolog a query is known as a goal, i.e. our goal was to find out if a dog could eat a cat. When we just have facts in our database and no rules then Prolog merely searches through the facts for a match. If a match occurs then our goal has been fulfilled and `Yes` is output. If no match is found then our goal fails and `No` is output.



Goal
A query that can result in either being fulfilled, in which case the result is Yes, or not being fulfilled, in which case the result is No.



GROUP TASK Activity
Write down two goals that will be fulfilled and two goals that will not. What do you think would be the result of the following goals?
`?-eat(pig,table)` `?-eat(dog,bird)`

Let us introduce a general fact into our program to say that everyone likes eating flies. Rather than having to write a long list of new facts, we can use a variable, say `X` to represent any value. Our new fact would be `eat(X,fly).` Variables must have an upper case first letter. Now if our goal is `?-eat(dog, fly)` or `?-eat(lion,fly)` or `?-eat(egg,fly)` the result will always be true.

Let us further generalise by introducing a rule into our system. Say that if a lion can eat a dog and a dog can eat a cat then a lion must be able to eat a cat. By continually applying this rule we see that all animals will be able to eat flies so we can remove `eat(X,fly)` from our facts. Generalising this statement we get something like this: If A can eat B and B can eat C then A can eat C, in Prolog we write this as:
`eat(A,C):-eat(A,B),eat(B,C).`
Notice that A, B and C are in upper case so they are variables. Facts and rules combine to form a knowledge base. During execution of a goal Prolog interrogates the knowledge base in an attempt to fulfil the goal.



Knowledge base
In terms of logical programming, a knowledge base is a database containing all the facts and rules.

As an example, we wish to know if dogs eat spiders. So we execute the goal `?-eat(dog,spider)` and it yields the result `Yes` as expected. How is Prolog executing this goal? Firstly, the list of facts is examined in search of a direct match; none is found. Next the rules are examined. In our example, only one rule exists. This rule allows us to reduce our goal into two sub-goals, `eat(dog,X)` and `eat(X,spider)`. We then examine `eat(dog,X)` and compare it to our facts. The first, and in this case the only match is `eat(dog,cat)`, the sub-goal `eat(dog,X)` is fulfilled. For our original goal to be fulfilled we require the sub-goal `eat(cat,spider)` to be fulfilled. Breaking this goal down using our rule, we get

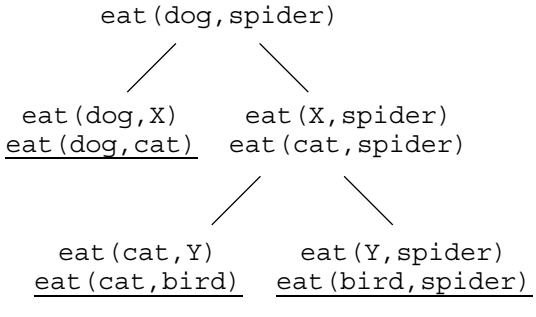


Fig 9.7
Tree showing a trace of the fulfilment of a goal.

`eat(cat,Y)` and `eat(Y,spider)`. Examining our facts, we find that `eat(cat,bird)` fulfils the first of these goals and `eat(bird,spider)` fulfils the second. As a result of these inferences, the original goal is fulfilled and `Yes` is output.

The inference engine carries out this processing. The inference engine is the processing unit of logical programming languages. Inference engines apply the knowledge contained within the knowledge base to reach conclusions about goals. They do this in an organised and systematic manner.



Inference engine
The control mechanism that applies knowledge, contained in a knowledge base, to resolve goals. The result being fulfilment or failure of the goal.



GROUP TASK Activity
Draw a trace tree, like the one in *Fig 9.7*, for each of the following goals.
`?-eat(lion,bird)` `?-eat(cat,fly)`

The goal `?-eat(bird,dog)` should result in an output of `No`. Actually, Prolog reports a stack overflow error. Let us examine the way Prolog's inference engine approaches the resolution of this goal. As `eat(bird,dog)` is not a fact within the knowledge base we split it into sub-goals using our rule. As a consequence we have `eat(bird,A)` and `eat(A,dog)`. We need to fulfil `eat(bird,A)`. There is a fact `eat(bird,spider)` that fulfils this goal. As `A` could be spider we now try to resolve `eat(spider,dog)`. This is not a fact so again we use our rule. We need to resolve `eat(spider,B)` and `eat(B,dog)`. We find a fact, `eat(spider,fly)` that fulfils the first of these sub-goals, so B could be fly. We need to resolve `eat(fly,dog)` which is not a fact so we once again use our rule resulting in `eat(fly,C)` and `eat(C,dog)`. No fact exists for `eat(fly,C)` so again we use our rule which gives us the sub-goals `eat(fly,D)` and `eat(D,C)`. Again, no facts for `eat(fly,D)` so we apply our rule giving us `eat(fly,E)` and `eat(E,D)`. This process continues infinitely and eventually causes a stack overflow error.

```
eat(lion,dog).
eat(dog,cat).
eat(cat,bird).
eat(bird,spider).
eat(spider,fly).
eat(A,C):-eat(A,B),eat(B,C).
```

Fig 9.8
Prolog code showing our knowledge base. Note that each fact or rule ends with a full stop.

This is a common problem encountered in Prolog when rules contain themselves. This is an example of a form of repetition called recursion. Some Prolog compilers and interpreters have mechanisms to detect these infinite inferences occurring, remember however, that logically they should occur.

The same problem can occur within imperative programs. A call to a procedure from within itself can often result in a stack error as the code attempts an infinite number of calls to itself. Care must be taken when using recursion.



Recursion
In terms of the logical paradigm recursion is a repetition mechanism where a rule contains itself.

A work-around is to create a new rule based on the old rule. In our example code, we'll call this new rule `caneat` where `caneat(dog,spider)` means a dog can eat a spider. Following is our corrected rule:

```
caneat(A,C):- eat(A,C).
caneat(A,C):- eat(A,B),caneat(B,C).
```

Let us examine our new goal question, `?-caneat(bird,dog).` The goal does not match any facts so we examine our two rules. Using the first rule we get `eat(bird,dog)`. This is not a fact and cannot be broken down. Examining our second rule gives us `eat(bird,X)` and `caneat(X,dog)`. The facts indicate that `X` could be spider, as `eat(bird,spider)` is a fact, in which case we need to resolve `caneat(spider,dog)`. Again, this is not a fact and using our first rule neither is `eat(spider,dog)`. Our second rule results in `eat(spider,Y)` and `caneat(Y,dog)`. `Y` could be fly as spiders eat flies, so we try to resolve `caneat(fly,dog)`. This goal expands to `eat(fly,Z)` and `caneat(Z,dog)`. So far the operations of the inference engine have been similar to our original example above. This is the point at which the addition of our new rule affects the operation of the inference engine. No fact or rule exists in our knowledge base to resolve `eat(fly,Z)` any more. As a consequence, the output is the coveted result `No`. Which means birds cannot eat dogs. Phew! All that work for such an obvious result.



GROUP TASK Activity
Work through the goal `?-eat(lion,cat)` to ensure our modified Prolog code still gives the correct output of `Yes`.



GROUP TASK Discussion
How has our modified code solved the problem? Surely our code is still logically the same as the original? Discuss.

There are two common strategies used to resolve goals; backward chaining and forward chaining. So far, our examples have used a backward chaining strategy. We started with a goal and sequentially searched through our knowledge base in search of facts and then rules that could be used to prove our goal. If we found a fact then the goal was proved. If we found a rule then we commenced our search once again at the start of the knowledge base using the goals within the rule. This process continued until we either proved our goal or reached a dead end.

Forward chaining uses almost the reverse strategy to backward chaining. When using a forward chaining strategy we need to know some initial facts are true. We use these facts to reach a conclusion. It is possible for forward chaining to result in more than one conclusion. Prolog is able to use a combination of backwards and forward chaining; it depends on the question and the facts and rules held in the knowledge base. Let us leave Prolog now and consider a conceptual example to illustrate both backwards and forwards chaining.



Consider the following:

We have the following set of rules in regard to aircraft that we know to be true:



Rule 1. If a plane has a jet engine and a single seat then it is a jet fighter

Rule 2. If a plane has a jet engine and a pressurised cabin then it can fly above 15000 feet

Rule 3. If a plane has fixed windows then it has a pressurised cabin

Rule 4. If a plane can fly above 15000 feet then it must be air-conditioned

Suppose we wish to prove using these rules, that a jet with fixed windows must be air-conditioned. Let us examine this problem using firstly a backward chaining strategy and then using a forward chaining strategy:

Backward chaining



Backward chaining
Assume the theory is true and then ask questions to systematically verify the necessary rules are present.

Using this method we commence with our desired goal, namely to prove that a jet with fixed windows must be air-conditioned. We scan down our rules until we find one that results in an air-conditioned plane. Rule 4 meets our needs. This rule provides us with a sub-goal to prove, namely that our plane can fly above 15000 feet. In other words, if we can prove our plane can fly above 15000 feet our supposition must be true. We now scan our rules looking for one that results in a plane that flies above 15000 feet. Rule 2 meets this requirement. Two new sub-goals result; the plane must have a jet engine and it must have a pressurised cabin. We know the plane has a jet engine, this was part of our question but does it have a pressurised cabin? Rule 3 proves it does; any plane with fixed windows has a pressurised cabin hence our plane must have a pressurised cabin because we know it has fixed windows. Our theory has been proved: jets with fixed windows must be air-conditioned.

Forward chaining



Forward chaining
Start from the beginning of the facts and rules and ask questions to determine which path to follow next to arrive at a conclusion.

Using this approach, we use the knowledge that our jet has fixed windows and attempt to reach a conclusion. Hopefully our conclusion will be that the plane must be air-conditioned. Scanning our list we find that Rule 3 means our plane must also be pressurised because it has fixed windows. We now know our plane is a jet with fixed windows that is pressurised. Using this information, we scan our rules again. We find that Rule 2 is resolved correctly, so our plane can fly above 15000 feet. As a consequence, Rule 4 becomes relevant and our plane must be air-conditioned. Our theory has once again been proved.



GROUP TASK Discussion
Forward chaining can lead to more than one conclusion. Why is this the case? Discuss. Use our aircraft rules to point out an example where this could occur.

Rather than continuing to explain the syntax and operation of Prolog in further detail a number of example HSC style questions have been reproduced below. Each of these questions illustrates different Prolog concepts and the suggested solutions include notes to further explain any new concepts. It would be worthwhile entering the source code within each question into Prolog and testing to confirm the answers and reinforce the ideas within your own mind. You cannot learn a programming language by reading a book so it is also important to learn to develop Prolog source code for some simple problems yourself.



HSC style questions:

Question 1.

Consider the following Prolog code:

```
member(A, [A|B]).   % Rule 1
member(A, [B|C]):- member(A, C).   % Rule 2
```

Note that the only solution for the goal [A|B]=[7, 3, 9, 4] is A=7 and B=[3, 9, 4]

(a) Explain how the goal member(5, [2, 5, 4]). would be evaluated.

(b) Explain how the goal member(6, [2, 5, 4]). would be evaluated.

(c) Explain how the goal member(X, [2, 5, 4]). would be evaluated.

Suggested solutions

(a) member(5, [2, 5, 4])
= member(5, [2 | 5, 4]) % matches rule 2
= member(5, [5, 4]) % new goal so we can prove rule 2
= member(5, [5 | 4]) % matches rule 1
Hence, goal evaluates to True.

(b) member(6, [2, 5, 4])
= member(6, [2 | 5, 4]) % matches rule 2
= member(6, [5, 4] % new goal to prove rule 2
= member(6, [5 | 4]) % matches rule 2 again
= member(6, [4]) % new goal to prove rule 2
= member(6, [4 |]) % matches rule 2 yet again, as empty list is still a valid list
= member(6, []) % new goal to prove rule 2, but does not match either rule
Hence, goal evaluates to False.

(c) member(X, [2, 5, 4])
= member(X, [2 | 5, 4]) % matches rule 1 when X = 2
Hence, goal is True when X=2
Continuing on...
= member(X, [2 | 5, 4]) % also matches rule 2
= member(X, [5, 4]) % new goal to prove rule 2
= member(X, [5 | 4]) % matches rule 1 when X=5
Hence, goal is also True when X=5
Continuing on...
= member(X, [5 | 4]) % also matches rule 2

= member(X, [4]) % new goal to prove rule 2
= member(X, [4 |]) % % matches rule 1 when X=4
Hence, goal is also True when X=4
Continuing on...
= member(X, [4 |]) % also matches rule 2
= member(X, []) % new goal to prove rule 2, but does not match either rule
Hence, goal evaluates to False
In summary, 3 solutions, namely X=2, X=5 and X=4.

Notes

- In Prolog square brackets are used to denote a list. A list is a data structure which simply groups a number of data items. The data items are separated using commas. Unlike an array the data items in a list may be of different data types. For example each of the following are legal Prolog lists [1, 2, 3, 4], [ant, 5, 6.87, wild, 'Sam Davis'] and []. The last example [] is known as the empty list.
- A vertical bar | is used to separate the head of a list from its tail. The head of a list is simply its first item and the tail is the remaining list. For example if we have the list [ant, bird, cat, dog] and we are matching a goal including [A|B] then A will match with the data item or atom ant and B will match with the list [bird, cat, dog]. In many ways Prolog is simply attempting to match the pattern of variables in the goal with its known facts.
- The percentage sign % is used to begin a comment within Prolog source code.
- Essentially the member predicate returns true if an item exists within a list. The predicate member is actually included in most versions of Prolog. Hence when testing this code you may need to use a different name for the predicate, such as my_member or similar.
- In part (c) a total of 3 solutions are found. When testing in Prolog the inference engine is looking to prove the goal is true. Therefore it needs only to find one possible solution. To cause Prolog to find additional solutions requires the inference engine to continue looking for answers. Commonly a semi-colon ; is entered to instruct the inference engine to continue. In Prolog the semi-colon is a logical "or" whilst the comma we have used previously between rule clauses is a logical "and" operator. It is rare to use the semi-colon within Prolog source code.
- Rule 1 is saying if A is the first item in a list then return true. Rule 2 is saying if A is not the first item in a list (i.e. A is not equal to B) then recursively see if A is a member of the list C.



GROUP TASK Investigation
What will occur if the goal includes a variable within the list. For example how will the goal member(3, [34, 56, X]) be evaluated?



GROUP TASK Practical Activity
Research and develop a rule which can be used to count the number of items in a list. For example, the goal listcount(X, [34, 56, 66]) is only true when X is 3 and the goal listcount(3, [34, 56, 66]) evaluates to true.

Question 2.

The following rule has been defined in Prolog:

```
dollar([A, B, C, D]):-
        member(A, [0, 1, 2]),
        member(B, [0, 1, 2, 3, 4, 5]),
        member(C, [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]),
        E is 50*A+20*B+10*C,
        E=<100,
        D is (100-E)/5.
```

(a) The goal `dollar([1,2,0,2])` evaluates as true. Explain the processing performed as this goal is evaluated.

(b) Determine all the values for `M` and `N` which make the goal `dollar([1,1,M,N])` true.

(c) Explain why the three `member` sub-goals are required within the rule.

(d) Contrast the above Prolog rule with imperative source code that produces the same results.

Suggested solutions

(a) Each sub-goal is tested in turn. A is 1 which is a valid member of the list [0,1,2] so true, B is 2 which is a member of the valid B list so also true, C is 0 which is part of the C list and similarly the D value of 2 is valid. E is then calculated as 50*1+20*2+10*0 = 90. 90 is less than 100 so this sub-goal is also true. Finally (100-90)/5 equals 2 which is the D value, so this sub-goal is true

(b) There are a total of 4 pairs of solutions. M=0, N=6 then M=1, N=4 then M=2, N=2 and finally M=3, N=0.

(c) When a goal does not include values then the inference must try to assert values. The inference engine systematically tries each value in the list of possible values for each variable. If no set of possible values are known then inferencing cannot continue as no possible values can be asserted. Without the 3 member clauses we would then have to find or prove the values for A, B, C some other way.

(d) Imperative code must explicitly describe each step required to determine values for the variables A, B, C and D. A different algorithm would be needed to find D given values for A, B and C. Another algorithm for finding A given B, C and D, etc. A further algorithm would be needed to check for valid A, B, C and D combinations. In Prolog the inference engine determines the steps to solve the problem using the rule. If a value is unknown it uses what it does know (such as that the value is a member of the appropriate list) in an attempt to determine a value that makes the complete rule true.

Notes

- The dollar predicate is determining all the different ways of making up one dollar using 50, 20, 10 and 5 cent pieces.
- The dollar predicate can also be used to determine if numbers entered for each coin correctly total one dollar. For instance in part (a) of the question one 50 cent plus two 20 cent plus two 5 cent pieces do indeed total one dollar.

- In addition if the number of some coins is specified it will determine the number of each remaining coin required to make a dollar (if possible). This is illustrated in part (b) of the question where we begin with one 50 and one 20 cent piece and need to determine all the different combinations of 10 and 5 cent pieces to total one dollar.
- It is common practice in Prolog to begin a new line for each clause within a rule. The commas between each clause are logical "and" operators hence all clauses within the rule must be true for the goal to be true.
- The Prolog keyword "is" is used to explicitly assign a value to a variable. This is to distinguish between an equals sign which is used to compare two items.



GROUP TASK Practical Activity
Imagine 2 cent and 1 cent peices are still available. Modify the dollar rule so it includes combinations which use 2 and 1 cent coins.

Question 3.

The following facts have been defined in Prolog.

```
feathers(finch).
feathers(parrot).
```

This means, both finches and parrots have feathers.

(a) Create further Prolog facts and rules based on the following information:
- All birds lay eggs and have feathers and wings.
- Bats, parrots and finches have wings.
- Echidnas, parrots and finches lay eggs.
- Bats and cows give birth to live young.
- Cows, bats and echidnas suckle their young.
- All mammals give birth to live young and suckle their young.

(b) Explain how the inference engine is able to determine all the different birds known to the system using the above facts and rules.

Suggested solutions

(a)
```
% Facts
feathers(finch). % Provided in question
feathers(parrot). % Provided in question
wings(bat).
wings(parrot).
wings(finch).
lay_eggs(echidna).
lay_eggs(parrot).
lay_eggs(finch).
live_young(bat).
live_young(cow).
suckle_young(cow).
suckle_young(bat).
suckle_young(echidna).
```

```
% Rules
birds(X):- lay_eggs(X), feathers(X), wings(X).
mammals(X): - live_young(X), suckle_young(X).
```

(b) The inference engine looks at the rule for a bird, and identifies that it is satisfied by those facts which are associated with all three of lay_eggs, feathers and wings. That is, it first looks for a value of X where lay_eggs(X) is defined as a fact. Echidna makes this true, it then looks for a fact to establish if echidnas have feathers - no such fact exists so we back track and look for another X that works with lay_eggs. We find parrot. We also find facts to establish that parrots have feathers and also wings. As all three facts are established we know a parrot is a bird. Similar processing occurs to establish that finches are also birds. Within the given facts the only ones that satisfy all three requirements are parrot and finch.

Notes

- The inference engine works through the facts and rules sequentially from top to bottom as it attempts to prove a goal. For this reason the order in which facts and rules appear has a significant effect on the processing performed. It is best to list all facts first followed by the rules.



GROUP TASK Activity
Research and write further rules which distinguish between the following three types of mammal - monotremes, marsupials and placentals.

Question 4.

The following Prolog facts and rules describe the management hierarchy within a company.

```
supervises(fred,jack).
supervises(mel,jeff).
supervises(fred,mary).
supervises(mary,luke).
supervises(peter,luke).
supervises(jeff,fred).
supervises(jack,sue).
supervises(jeff,peter).
supervises(luke,dave).
supervises(dave,ron).

boss(A):-
    supervises(A,B),
    supervises(B,C),
    supervises(B,D),
    C\=D.
bigboss(A):-
    boss(A),
    not(supervises(_,A)).
```

The first fact supervises(fred,jack) means fred supervises jack.

(a) Explain how the goal bigboss(A). would be evaluated.

(b) Contrast the above logic paradigm solution for finding bigboss with an imperative paradigm solution to the problem.

Suggested solutions

(a) Steps during the evaluation of bigboss(A) include:

- First find a person that makes boss(A) true.
- This leads to the subgoal supervises(A,B) which of course is true for each fact in the database. As supervises(fred,jack) is first in the list then A binds to fred and B binds to jack.
- The next subgoal is then supervises(jack,C) which matches with supervises(jack,sue) so C binds to sue. The next subgoal is the same so D also binds to sue.
- The final subgoal within boss(A) is C\=D. In this case both C and D are bound to sue so this subgoal is false. We backtrack to the previous subgoal but there are no other facts where jack supervises somebody. Therefore the call to boss(A) is false when A is fred.
- We now begin the boss(A) call again. This time we look at the next (second) fact in our database, namely supervises(mel,jeff) therefore A binds to mel and B binds to jeff.
- The next subgoal is then supervises(jeff,C) which matches with supervises(jeff,fred) so C binds to fred. The next subgoal is the same so D also binds to fred. This doesn't work as C and D are the same so we backtrack one step and find the fact supervises(jeff,peter). This means D binds with peter and our C\=D clause is now true. Therefore mel is a boss as boss(mel) is true.
- We now need to show the second clause of bigboss(mel) is true. That is, show not(supervises(_,mel)) is true. Scanning down the facts there is no fact where mel is the second item. Therefore supervises(_,mel) is false which means not(supervises(_,mel)) is true.
- Finally, bigboss(mel) is true as both parts of the rule are true for mel. Therefore the goal bigboss(A) is found to be true.

(b) The logic solution describes the data and rules but makes no attempt to explain how to process these facts and rules to arrive at a result. An imperative solution must logically and unambiguously describe the sequence of processing steps needed to find the result. In effect Prolog's inference engine works out the processing steps, whilst an imperative solution needs to clearly state all steps.

In this example an imperative solution would first need to include a loop and decision to search through the data looking for people who have no supervisor. If such people are found then another loop and decision is needed to search for the people they supervise – the Bs in the Prolog version. Any Bs from this search are then used in turn as criteria for another search to see if the person supervises at least two other people – yet another loop and decision. The final imperative solution would include many lines of code compared to the two rules needed in the Prolog solution.

Furthermore the Prolog solution can do much more than simply find bigboss. It can be used to check if a person is a bigboss and it can also locate each boss, check if someone is a boss, find someone's supervisor, or who someone supervises. In an imperative language determining each of these things would involve major code modifications.

Notes

- The operator \= means does not equal. Therefore C\=D is true when the variables C and D are not bound to the same data values.
- The underscore character _ in Prolog is known as the anonymous variable. It is similar to other Prolog variables however its value is not reported. In addition if multiple anonymous variables are used within a Prolog statement then each is presumed to be distinct. For example, we could develop a rule called super to find people who are both supervisors and who are also supervised. We could write super(A):- supervises(A, _), supervises(_, A). In this rule the two anonymous variables are not linked and hence can hold different values. For instance super(fred) is true because supervises(fred, jack) and supervises(jeff, fred) are both true together despite jack and jeff being different. In essence the rule super(A):- supervises(A, _), supervises(_, A). is logically the same as super(A):- supervises(A, B), supervises(C, A).
- In many references predicates are quoted along with the number of arguments they accept. For example in the above question we have supervises/2, boss/1 and bigboss/2. It is important to specify the number of arguments because it is possible to use the same name for predicates with a different number of arguments. For example we could have prd/2, prd/3 and prd/4 which perform different functions.



GROUP TASK Research Activity
Write a predicate which finds all the employees known to the system who are not supervisors. Try to combine them within a list data structure (hint: research and use the built-in findall predicate).

APPROPRIATE USE OF THE LOGIC PARADIGM

Prolog is the leading programming language for artificial intelligence (AI) applications and research. The logic paradigm is based on formal predicate logic which has historically been the basis of much research into how humans think, reason and make decisions. In Prolog and logic we describe the problem we wish to solve rather than describing how to solve the problem. This is a declarative approach to programming rather than the more common procedural and imperative approach used by most programming languages. A declarative approach allows a single program to be used to solve a variety of related problems.

The logic paradigm is particularly good at representing knowledge and then using this knowledge to reason. We have seen many examples of knowledge representation as we studied Prolog and we have described the reasoning used by Prolog's inference engine. Often the reasoning performed by the inference engine is essentially pattern matching. Even in our simple examination of Prolog we've seen how a predicate such as parent(Parent, Child) matches with the fact parent(joe, sue) when Parent is joe and Child is sue. This pattern matching ability is used for a variety of AI applications including natural language processing such as grammar and spell checks. It is also used to simulate the reasoning of human experts within expert systems.

Expert systems

An expert system is used to perform functions that would normally be performed by a human expert in that field. A doctor may use an expert system to diagnose illnesses or economists may use an expert system to forecast economic trends. In general, expert systems do not reach definite conclusions rather they weigh up the evidence and present possible conclusions. Expert systems are designed by knowledge engineers who specialise in understanding how a human expert reasons. The reasoning is stored in a knowledge base that is interrogated by an expert system shell. Expert system shells provide the framework for an expert system to which specialised knowledge must be added by knowledge engineers.

Expert systems use facts and rules to reach conclusions using similar techniques to those used in Prolog. Many expert systems also allow the inclusion of heuristics. These are rules that are generally accepted as true within the particular specialist area. Heuristics are often described as rules of thumb. Heuristics are what give expert systems a human feel. As humans, we reach conclusions using a variety of mechanisms. Some of them are precise but many are not. Simulating the experience of a human expert using software is a difficult and data intensive process. Many decisions we make in every day life are not based on strict facts and rules rather they are based on creative original thought and feelings. At present, computers are not able to accurately simulate this type of decision-making.



Heuristic
A rule of thumb often based on experience. It usually results in one or more possible solutions.

It is difficult to discuss expert systems without mentioning neural networks. Neural networks are different from expert systems in that they are able to learn. This new knowledge is then available in the future to help reach new conclusions. Expert systems cannot learn new information. Neural networks are composed of neurons that operate in a similar fashion to the biological neurons within our brains. Each neuron has a number of inputs and a single output. Each input is given a particular weighting within the neuron. When the sum of the weightings reach a particular threshold then the output fires. Each output is linked to one or more inputs to further neurons within the system. Our brain contains billions of biological neurons that are connected in intricate and complex ways; neural networks are similarly complex. Neural networks can operate heuristically; the weightings of each input to a neuron changing with experience and different circumstances.

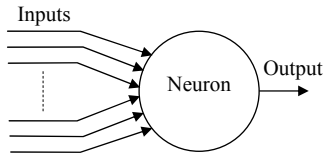


Fig 9.9
Neurons are the basis of neural networks.



GROUP TASK Discussion
Expert systems and neural networks are not the same. Discuss the similarities and differences. Does the human brain operate more like a neural network or an expert system?

An expert system shell is a software product that can be used to create an expert system. Facts, rules and probabilities of events occurring are entered into a knowledge base. The expert system shell then interrogates the knowledge base to reach conclusions. Often the conclusions are reported in terms of probabilities e.g. given the data in a certain knowledge base an expert system shell calculates the probability of being struck by lightning as 0.00001, however if we tell the system we are outside in a thunderstorm then the probability of being struck is 0.00035. Altering the knowledge base will influence the final conclusions. Probabilities are usually expressed as values between 0 and 1. 0 means it will never occur and 1 that it will definitely occur.



Consider the following:

An expert system has been developed using the expert system shell 'Hugin Lite 5.6'. This expert system is used to calculate the probability of contracting a hereditary disease. People can either suffer from the disease, be carriers of the disease or be free of the disease.

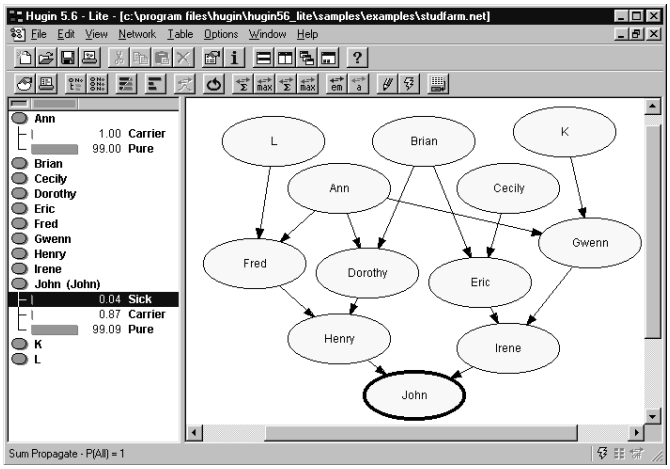


Fig 9.10
GUI for the expert system shell Hugin Lite 5.6.

Fig 9.10 shows the graphical user interface for this expert system shell, with a knowledge base about a particular disease loaded. Given the present state of the knowledge base, John has a 0.04% chance of contracting the disease, 0.87% chance of being a carrier and a 99.09% chance of being pure in regard to the disease.

Let's say we determine that Ann, who is John's great-grandmother, was definitely a carrier. This means that Ann's probabilities require alteration so that the probability of being a carrier is 100%. As a consequence, John is 2.66% likely to suffer from the disease, 28.28% likely to be a carrier and 69.06% likely to be disease free. *Fig 9.11* shows the outputs after this change.

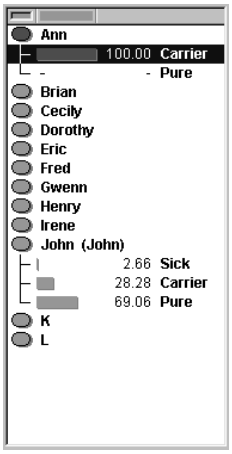


Fig 9.11
Outputs after modifying the knowledge base.

In this example, we are viewing a GUI for the expert system shell. This product utilises an application program interface (API) that allow programmers to access the functions of the expert system shell from within their own programs.



GROUP TASK Investigation
Download a demonstration version of an expert system shell from the Internet. Examine some of the sample knowledge bases. Can you create a knowledge base similar to our Prolog animal example using this shell?

SET 9B

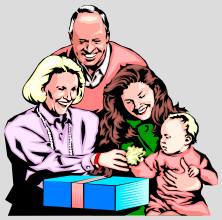
1. In terms of logical programming, what is the name of a database that contains facts and rules?
   (A) Logical base.
   (B) Knowledge base.
   (C) Fact base.
   (D) Query base.
2. The processing unit of logical programming languages is called
   (A) the control mechanism.
   (B) the goal fulfilment.
   (C) the inference engine.
   (D) the knowledge base.
3. Expert system shells provide the framework for
   (A) neural networks.
   (B) expert systems.
   (C) hierarchal systems.
   (D) open database systems.
4. Dorothy is investigating her family tree and wishes to find out who her great, great uncle is on her mother's side. Which strategy would be most suited to Dorothy's investigation?
   (A) Backward chaining.
   (B) Forward chaining.
   (C) Sequential testing.
   (D) Goal fulfilment.
5. Theobald is testing a theory and has started by assuming the theory is true. Which strategy is Theo most likely using to resolve this goal?
   (A) Forward chaining.
   (B) Goal fulfilment.
   (C) Backward chaining.
   (D) Sequential testing.
6. Heuristics are often described as
   (A) rules of thumb.
   (B) human feelings.
   (C) creative thoughts.
   (D) decisions.
7. The following is a fact written using Prolog syntax. jump(horse,fence). Which part of the syntax is called a predicate?
   (A) jump.
   (B) fence.
   (C) horse.
   (D) The entire syntax.
8. Which paradigm is more likely to be used in the area of artificial intelligence?
   (A) Imperative.
   (B) Object oriented
   (C) Functional.
   (D) Logic.
9. A repetition mechanism where a rule is contained within itself, is called
   (A) recursion.
   (B) a stack overflow.
   (C) an inference mechanism.
   (D) a call repetition.
10. Walter is employed by a computer company that develops financial models for the stocks and bonds industry. It is Walter's responsibility to add to a knowledge base contained within the system. Which type of system would Walter most likely be working with?
    (A) An expert system.
    (B) A hierarchal system.
    (C) An expert system shell.
    (D) A relational system.

11. The logic paradigm is based on facts and rules. These facts and rules are used to test theories and find conclusions. In logical paradigm terms what are facts and what are rules? Explain the difference between facts and rules?

12. Expert systems include knowledge bases and inference engines. What is a knowledge base and what is an inference engine? How do these two components combine to form an expert system?

The relationships between family members can be represented as a knowledge base. Examine the relationships existing in the following scenario. Use this scenario to answer questions 13, 14 and 15.

- Bob has three sisters, Mary, Alice and Louise.
- Bob's parents are Ted and Margo.
- Margo's parents are John and Sue.
- Margo has two brothers, Mike and Tom.
- Tom's parents are Fred and Wilma.
- Mike has two sons, Dave and Paul.



13. Develop a list of facts using Prolog syntax, to represent each of the above relationships. For example sibling(bob,mary) means Mary is Bob's sibling. Make sure to include facts in regard to the sex of each family member eg. female(margo) and male(bob).

14. Family relationships adhere to strict rules. For example, to be somebodies grandmother you must be female and your child must be the parent of the grandchild. This could be expressed as the Prolog rule grandmother(A,B):-female(A),mother(A,C),parent(C,B). Write rules for each of the following relationships:
    - Mother
    - Father
    - Sister
    - Brother
    - Cousin
    - Aunt
    - Uncle
    - Parent
    - Grandmother
    - Grandfather

15. Use your knowledge base to determine the following:
    - Use a backward chaining strategy to confirm that Margo is Dave's aunty.
    - Use a forward chaining strategy to find all of John's grandchildren.

OBJECT ORIENTED PARADIGM

Earlier in this chapter we introduced a number of general concepts underpinning the object oriented paradigm. We discussed objects as self-contained units, each of these units containing data elements, or attributes, and methods. We now expand our object oriented paradigm as it is applied in most object oriented programming languages. The term object oriented programming is often abbreviated to OOP in many references; similarly object oriented is abbreviated using the acronym OO.

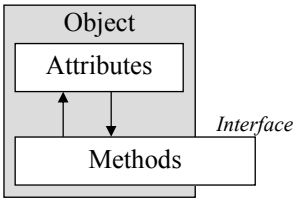


Fig 9.12
Diagrammatic view of an object.

Object oriented languages are fast becoming the most popular of all languages. The reusability of the code makes them an excellent choice for a large variety of applications. Many traditional imperative languages have been updated by adding object oriented functionality. For example, C has been modified to create C++, Delphi is similar to Pascal with OO capabilities. There are even versions of Prolog that include OO functionality. The first OO language was Simula, developed in 1967, following came Smalltalk in 1972, which is still in common usage today.

One common misconception is that event-driven and object oriented are interchangeable terms. Event-driven is a term used to describe programs that react to certain conditions e.g. clicking on the OK button on a form causes a particular module to be executed or selecting a menu item invokes a particular function. Programming languages designed to create these GUI elements are called event-driven languages. It is true that most of today's event driven languages are also object oriented languages, however this is not necessarily the case. This will become clearer as we examine further OO concepts. Perhaps the most commonly used OO languages are Java and C++. Bjarne Stroustrup originally developed C++ in 1986 whilst Java was first developed in the early 1990s by engineers at Sun Microsystems. The syntax of both C++ and Java, at first glance, appears similar; however there are a number of important differences. Java is not a subset of C++ and neither is C++ a subset of Java, however the two languages do have similarities in regard to general syntax. Our examples that follow will be written using Java.

As we did in the previous section, when we examined the logic paradigm, we will introduce new OO concepts as we develop a design for a program.



Consider the following:

We wish to write a computer adventure game. We'll call our game *The Good, the Bad and the OO*. Our game has two distinct types of characters, goodies and baddies. The goodies are the players and the baddies are automated things that try to hinder and fight the goodies. The aim is to collect all the treasure without the baddies wiping you out (the OO wins if a runtime error is encountered!) Baddies can take a number of forms. They can be monsters or rooms or even trees. The important thing is that baddies are able to attack goodies somehow e.g. a monster might be able to eat a goodie, the room may be able to collapse and a tree may be able to drop coconuts on a goodie. Goodies move through a maze of streets, parks and buildings trying to collect treasure. As a goodie, you fight baddies encountered on the journey. The baddies fight back to hinder the goodies progress.

Firstly, we need to design objects that will form the basis of our program. To do this we need to understand what an object is. Objects contain attributes and methods. Attributes are the stuff an object knows and remembers; methods are actions the object can do. Attributes contain the data known by an object. Attributes can only be accessed and altered by the object's methods. In other words, an object's methods provide an interface through which the object communicates with other objects.

In our adventure game we'll need an object for each goodie and each baddie in the game. Let's consider a particular goodie for a moment. We call this goodie MrNice. He's a timid character who is not particularly agile but is very intelligent and quite healthy. MrNice needs to know how to move around, get treasure and fight monsters. *Fig 9.13* shows the contents of MrNice represented as an object. As a result of MrNice's methods, his attributes can change e.g. if he doesn't do too well in a fight his health will suffer. If he gets some treasure he becomes more outgoing and his health improves. The only way MrNice's attributes can change is through his methods.

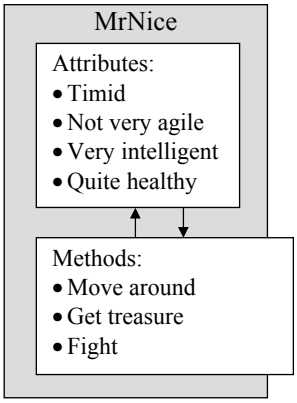


Fig 9.13
Diagrammatic view of MrNice.

If we examine the MrNice object, we notice that his attributes are generally common to all the goodies in our game. All our goodies will require a personality (MrNice is timid), agility, intelligence and health. Similarly, all goodies need to move around, get treasure and fight baddies. We can design a class called goodie that describes a general goodie. We can then use this class to create individual goodies for use in our game.

To create the goodie class, we'll need to be a little more abstract than we were with our description of MrNice. Let's say each attribute is represented using an integer from 0 to 10, where 0 is low and 10 is high. Our integer values have the meanings described in the table shown in *Fig 9.14*. MrNice starts the game with the following attribute values: personality=3, agility=4, intelligence=8, health=7. If he gets in a fight with a baddie and the baddie gives him a rough time then his health might go down to 5. If he outwits a baddie during a fight or gets some hard-to-find treasure his intelligence may increase to 9.

Attribute	Scale of values		
	0	to	10
personality	Extremely timid	to	really extroverted
agility	Uncoordinated	to	superb athlete
intelligence	Stupid	to	genius
health	Dead	to	perfect

Fig 9.14
Table of attribute values for a goodie.



Encapsulation
The process of hiding an object's data and processes from its environment. Only the object can alter its own data.

The values of MrNice's attributes can only change as a result of one of his methods altering the attribute. This concept is known as *encapsulation*, MrNice's attributes (or data items) are private and are contained within him. In object oriented terms, *encapsulation* means that objects control their own private attributes using their methods. No other object (or class) can alter another objects attributes directly rather they must use the objects public methods. Encapsulation is a vital concept in OOP. It allows the creation of robust and reusable classes of objects. As programmers, we know that if an attribute has changed then the object must have made the change itself; a great help when testing and debugging programs.

Let us now return to our creation of a goodie class. Declaring a class is in many ways similar to declaring a record as a new data type in a procedural language. For example, a record may be defined with three fields, surname of type string, age of type integer and sex of type Boolean. A particular instance of this record could be created called Frank and another called Mary. Frank contains the fields Frank.surname, Frank.age and Frank.sex. Similarly, Mary contains Mary.surname, Mary.age and Mary.sex. In OOP, these definitions are called classes and a particular instance of a class is called an object. In our game we'll need a class for goodies and another class for baddies. We'll consider the design of the goodie class in detail. We use the goodies class to create an instance of a goodie, e.g. our MrNice character. The class definition tells MrNice what attributes he has and how to accomplish his methods.

In Java we could declare our goodie class with the following code:

```
public class Goodie {
    private int personality;
    private int agility;
    private int intelligence;
    private int health;
    public void move() {
    }
    public void fight() {
    }
    public void getTreasure() {
    }
}
```



Class
The definition of a category of objects. The class defines all the common attributes and methods of the different objects that belong to it.

This is a perfectly good class declaration for a goodie, (see note below). The problem is objects created using this class can't really do anything. We haven't described how a goodie moves, gets treasure or fights baddies. As a consequence there is no way we can alter the goodies attributes. (Remember the attributes are encapsulated within each object). Secondly, we need to be able to initialise the attributes to their starting values. We'll consider these problems later, first let's create some objects using the Goodie class. Creating an object based on a class is called *instantiation*, meaning we are creating a particular 'instance' of the object that can be used in our program. The following Java statements instantiate the three objects MrNice, MissLovely and MrWild:

```
Goodie mrNice;
mrNice = new Goodie();
Goodie missLovely = new Goodie();
Goodie mrWild = new Goodie();
```

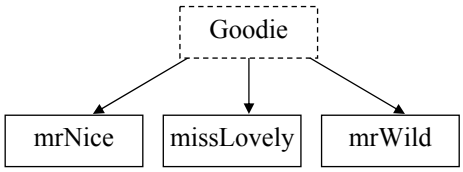


Fig 9.15
The goodie class is used to produce three objects; mrNice, missLovely and mrWild.

The object mrNice has been created using two lines of Java code. The first line creates a variable called mrNice of type Goodie. The second line instantiates mrNice by creating a new Goodie object and assigning it to the variable called mrNice. When instantiating missLovely and mrWild these two processes have been combined in a single line of code.

To enable us to describe classes and objects diagrammatically we'll use dashed rectangles for classes and normal rectangles for objects. *Fig 9.15* shows the Goodie class which has been used to create our three objects.

Note: In Java int is used to declare each of the attributes with an integer data type. void simply means there is no return data type for this item. The double brackets tell Java that this is a method, parameters for the method would be listed within these brackets. Curly brackets are used to enclose statements, such as the class statement in this example and also the statements describing how each of the methods will be performed. It is common practice for class names to be capitalised whilst other identifiers begin with a lower case letter.



GROUP TASK Discussion
We compared declaring a class in OOP to declaring a new data type in procedural or imperative programming. What are the differences between these two types of declarations? Discuss.



GROUP TASK Discussion
Take a break from our adventure game and make sure you understand the following terms: object, attribute, method, class, encapsulation and instantiation. Discuss each term.

So far we have only created goodies, our program requires some baddies as well. Rather than go through the design of the baddies class in detail we'll just reproduce it below:

```
public class Baddie {
    private int agility;
    private int health;
    private int aggression;
    private int badType;
    public void move() {
    }
    public void fight() {
    }
    public void makeNoise() {
    }
}
```



Inheritance
The ability of objects to take on the characteristics of their parent class or classes. Inheritance encourages modularity and robust code.

This baddie class declaration is similar in many respects to the goodie class declaration. The attributes are the same except intelligence has been removed and aggression and badType have been added. A makeNoise method has also been added. The move and fight methods would essentially be the same as the move and fight methods contained within the goodie class. It would be nice if we could somehow combine these class declarations so we don't need to reproduce and then test almost identical segments of code.

What if we create a super class called character? This super class could contain all the common attributes and methods used in both goodie and baddie classes. This leads us to a new concept central to OOP, *inheritance*. Subclasses inherit all the attributes and methods of their super class. In Java terminology the subclass is said to extend the super class. In many references subclasses are called child classes or derived classes and super classes are called parent classes. The ability of a class to inherit features from other classes is extremely useful in OOP. Because each class has been developed as a robust unit we know that inherited components will also be robust. Development of new objects with similar characteristics to existing objects is greatly simplified using inheritance.

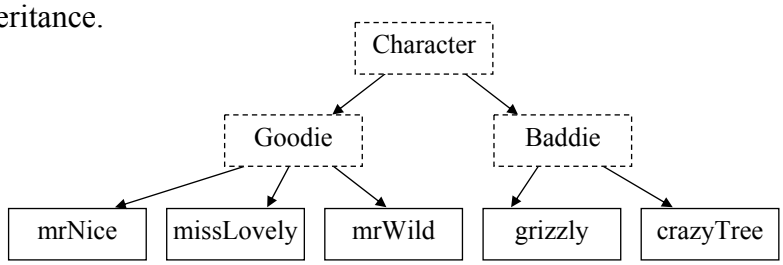


Fig 9.16
The goodie and baddie classes inherit all the attributes and methods contained in the character class.

Following are our amended declarations in Java:

```
public class Character {
    private int agility;
    private int health;
    public void move() {
    }
    public void fight() {
    }
}
public class Goodie extends Character {
    private int personality;
    private int intelligence;
    public void getTreasure() {
    }
}
public class Baddie extends Character {
    private int aggression;
    private int badType;
    public void makeNoise() {
    }
}
```

In some OOP languages it is possible for a class to inherit the characteristics of a number of super classes. This is termed multiple inheritance. For example, our CrazyTree object could have been created using a class that inherits from our baddie class as well as from a class that describes how trees in general behave.

The process of designing objects by breaking them down into component classes allows us to concentrate on the details of our current work. This is the process of *abstraction*. The hierarchy of classes is designed in such a way that each class is reduced so as to include only its necessary attributes and methods. Abstraction allows us to isolate parts of the problem and consider its solution apart from the main problem. Encapsulation and inheritance greatly assist in the process of abstraction. They allow us to put the overall problem aside with confidence whilst sub-problems are dealt with.



Abstraction
The process of designing classes so they are reduced to their necessary attributes and methods.



GROUP TASK Activity
Write down Java statements to declare each of our three goodies and two baddies shown in *Fig 9.16*. Given each character's name, what do you think the initial values of their attributes should be?



GROUP TASK Discussion
Explain the meaning of the terms inheritance and abstraction. How are these terms linked in OOP? Discuss.



GROUP TASK Research
It is possible to design abstract classes to prevent objects being instantiated from that class. Often super classes are abstract classes. Research to determine the reason for creating abstract classes.

Earlier we identified two problems with our class definitions. One of these problems was that when a new object is created we want to be able to initialise it by setting the value of its attributes. The second problem was to describe what happens when one of an object's methods is executed. We'll consider the initialising problem first and as part of the discussion we'll create some simple methods.

In OOP you cannot just assign values to attributes. For example you can't say mrNice.health=7, this contradicts our encapsulation concept. In Java encapsulating attributes within objects is achieved by specifying that each attribute is private. This prevents any changes to the attribute from outside the class. Only the class' methods can change the value of attributes. To initialise the value of an object's attributes requires a special type of method called a *constructor*. Every time an object is *instantiated* (created) its constructor method is executed. If you haven't written a constructor then the compiler creates a dummy one for you. This is fine if you don't really care what values are initially stored in an object's attributes. In our adventure game, we want to be able to create goodies with different initial characteristics. Similarly, we want to create baddies with different characteristics.

If you want a method to do something, which is usually the case or there is little reason for the methods existence, then you need to write a function definition for the method. These function definitions are the instructions that perform the processing within your object oriented code and are much like the imperative code we have learnt throughout this course. Say we want all our baddies to be created with both agility and health set at 5. We also want to be able to create baddies of different badType with different amounts of aggression. Constructor methods always have the same name as the class in which they reside. This is what tells the compiler that the method is a constructor rather than a normal method. Following are our class declarations with constructors that perform this task:

```
public class Character {

    private int agility;
    private int health;

    public Character(int a, int h) {
        if (a<0 || a>10)
            a = 0;
        this.agility = a;
        if (h<0 || h>10)
            h = 0;
        this.health = h;
    }
    public void move() {
    }
    public void fight() {
    }
}
```

```
public class Baddie extends Character {

    private int aggression;
    private int badType;

    public Baddie(int a, int b) {
        super(5, 5);
        if (a<0 || a>10)
            a = 0;
        this.aggression = a;
        if (b<0 || b>10)
            b = 0;
        this.badType = b;
    }
    public void makeNoise() {
    }
}
```

Baddie crazyTree = new Baddie(3, 2) would create a baddie object called crazyTree with agility and health values of 5, an aggression value of 3 and a badType of 2.



GROUP TASK Discussion

What will occur if the statement Baddie crazyTree = new Baddie(20,2) is executed? Based on this example discuss how encapsulation and inheritance helps to create robust reusable code.

The Java keyword this is used to specify the current object. The keyword super is used to refer to public items in the super class. As constructor methods do not have specific names then using the keyword super on its own refers to a constructor in the super class. Methods within the super class could also be executed from within a method in the subclass using the super keyword. For example to execute the move() method in the Character super class from within a method in the Baddie class we would write super.move().

Our constructor for the baddie class is sufficient for our needs, however we'll often need to create goodies with agility and health equal to 5 and also goodies where all attributes have different initial values. We add two constructors to the goodie class which pass on the agility and health inputs to the character class constructor.

```
public class Goodie extends Character {

    private int personality;
    private int intelligence;

    public Goodie(int a, int h, int p, int i){
       super(a, h);
       if (p<0 | | p>10)
           p = 0;
       this.personality = p;
       if (i<0 | | i>10)
           i = 0;
       this.intelligence = i;
   }
    public Goodie(int p, int i){
       this(5, 5, p, i);
    }
    public void getTreasure(){
    }
}
```

The statement Goodie mrNice = Goodie(4, 7, 3, 8) creates a goodie object called mrNice where agility=4, health=7, personality=3 and intelligence=8. Don't be too concerned about the detail of the Java syntax. In examinations we are more concerned about understanding the concepts of OOP rather than learning a particular language.



GROUP TASK Discussion
Why aren't constructors required when using imperative languages? What techniques are used to perform a similar function to constructors in imperative languages? Discuss.



GROUP TASK Discussion
Each class could also contain a destructor. In Java destructor methods are called finalize() and in C++ using a tilde "~" followed by the class name. What do you think is the purpose of a destructor?

Examine the goodie class definition above. To those of us familiar with imperative languages, the fact that the goodie class now contains two methods with the same name should be of concern. i.e. we have two methods (or functions) called Goodie within the goodie class definition. Although in this example the Goodie methods are constructors it is possible for any method to have multiple definitions within a single class and not only is it possible it is also very common. In object oriented programming this is a valuable and deliberate inclusion in all OO languages. The

compiler recognises different methods and functions not only by their names but also by the number of parameters and the type of those parameters. This concept is known as *overloading*. In our Goodie class the first version of Goodie contains four integer parameters a, h, p and i whereas the second contains just two integer parameters named p and i.



Overloading
Methods which have the same name but accept a different number of parameters or parameters of different types.

Let us expand our concept of overloading by considering how an object's methods are described within a program. We'll also learn how we can get our characters to actually perform their methods. Let's consider the getTreasure method which is part of each of the goodie characters. Usually when a goodie gets some treasure he feels better and more confident and as a consequence, his health and personality are both incremented by 1. Some treasure items scattered throughout the game are nasty; they can cripple a goodie causing their agility to be set to 0.

Within the Goodie class we write two different versions of our getTreasure method. Both versions have a single parameter however one is an integer and the other a char. Here's our first attempt:

```
public class Goodie extends Character {

     private int personality;
     private int intelligence;

    ...

    void getTreasure(int a) {
        super.health++;        //this line causes an error
        this.personality++;
    }
   void getTreasure(char a) {
        super.agility = 0;        //this line also causes an error
   }
};
```

Two errors occur as we are unable to access the health and agility attributes from within the Goodie class. This is because they are encapsulated within the parent Character class. We need methods within the parent Character class which allow us to set the value of these attributes. We note that the data validation logic required is the same as was used in the Character class constructor; hence some reworking of the code is performed to reduce repetition and improve maintainability of the code. Later we would like to have an array of characters so it makes more sense to store the name of each character as an attribute of each object. In addition a simple print method is written to assist with testing. The updated character class is as follows:

```
public class Character {
   private String name;
   private int agility;
   private int health;
   public Character(String n, int a, int h) {
      this.name=n.trim();
      this.setAgility(a);
      this.setHealth(h);
     }
```

```
    public void setAgility(int a){
        if (a<0 | | a>10)
            a = 0;
        this.agility = a;
    }
    public int getAgility(){
        return this.agility;
    }
    public void setHealth(int h){
        if (h<0 | | h>10)
            h = 0;
      this.health = h;
    }
    public int getHealth(){
        return this.health;
    }
    public String getName(){
        return this.name;
    }
    public void move() {
    }
    public void fight() {
    }
    public void print(){
        System.out.println("My name is "+this.name);
        System.out.println("Agility = "+this.agility);
        System.out.println("Health = "+this.health);
    }
}
```

The Goodie class is also updated to include more robust getTreasure() methods and to add a print() method to assist with testing:

```
public class Goodie extends Character {
    private int personality;
    private int intelligence;
    public Goodie(String n,int a, int h, int p, int i){
        super(n,a,h);
        if (p<0 | | p>10)
            p = 0;
        this.personality = p;
        if (i<0 | | i>10)
            i = 0;
        this.intelligence = i;
    }
    public Goodie(String n, int p, int i){
        this(n, 5, 5, p, i);
    }
    public void setPersonality(int p){
        if(p<0 | | p>10)
          p = 0;
        this.personality = p;
    }
```

```
    public void getTreasure(int a){
        int h = this.getHealth();
        this.setHealth(h++);
        int p = this.personality;
        this.setPersonality(p++);
    }
    public void getTreasure(char a){
        this.setAgility(0);
    }
    public void print(){
        System.out.println("Goodie");
        super.print();
        System.out.println("Personality = " + personality);
        System.out.println("Intelligence = " + intelligence);
    }
}
```

Suppose someone is playing our adventure game and they are controlling MrNice. They come across a particular piece of treasure so they press the appropriate key. Somewhere in your code the command mrNice.getTreasure(188); gets executed (the value 188 could have been any integer). This statement sends a message to the getTreasure method of the mrNice object. Because 188 is an integer rather than a character, MrNice's health and personality are incremented. If the command had been MrNice.getTreasure("k") then MrNice's agility would have been set to 0. The compiler sees the two getTreasure functions as different because their parameters are of different types; one is an integer (int) and the other a character (char). When using goodie objects to code other parts of our game, we just view the getTreasure method as a single method; we say that getTreasure is *overloaded*.



Overriding
Methods (or properties) which replace an inherited method (or property).

Overriding is a further OOP concept related to overloading. Overriding occurs when a subclass includes a different version of a method (or attribute) which it inherited from its parent class. The version within the subclass replaces or overrides the inherited parent version. Overriding only occurs when the number and type of parameters is the same in both versions. Overriding allows the programmer to change the behaviour of subclass objects compared to the behaviour of the parent class. For example, our Character class contains a move() method which is currently inherited by both goodies and baddies. If we decide we'd like baddies to move differently then we can write another move() method within the baddies class. This means that whenever a baddie moves the code within the baddie class will execute. However when goodies move the original inherited move() code from the parent Character class will continue to be executed.



GROUP TASK Discussion
Examine the above class definitions to locate and distinguish between examples of overloading and of overriding.



GROUP TASK Activity
Determine the output from the following lines of code:

```
Character myGoodie = new Goodie("Luke", 6, 8);
myGoodies.getTreasure(123);
myGoodie.print();
```

Changes are also made to the Baddie class. The updated Baddie class definition is reproduced below:

```
public class Baddie extends Character {
   private int aggression;
   private int badType;
   public Baddie(String n, int a, int b) {
      super(n, 5, 5);
      if (a<0 | | a>10)
         a = 0;
      this.aggression = a;
      if (b<0 | | b>10)
         b = 0;
      this.badType = b;
   }
   public void makeNoise() {
   }
    public void print(){
      System.out.println("Baddie");
      super.print();
      System.out.println("Aggresion = "+aggression);
      System.out.println("BadType = "+badType);
   }
}
```



GROUP TASK Discussion

Currently code similar to the following lines are repeated in each of the Character, Goodie and Baddie classes:

```
if (a<0 | | a>10)
    a = 0;
```

Suggest modifications so this code only appears once.



GROUP TASK Activity

Currently if the new value for an attribute is outside the range 0 to 10 the attribute is set to 0. Alter the code so that new values less than 0 cause the attribute to be set to 0 and new values greater than 10 cause the attribute to be set to 10.

The final OOP concept we need to discuss is *polymorphism*. Generally *polymorphism* is the ability to appear in more than one form. Poly means many and morph refers to different forms of an object. A butterfly can be described as polymorphic; it was once in a different form, that of a caterpillar and before that it was an egg. The butterfly is still the same species of insect; it just takes on various different forms or morphs.

In terms of OOP, *polymorphism* refers to the ability of the same command to process objects from the same super class differently depending on their subclass. At runtime the system chooses the precise method to execute based on the subclass of each particular object being processed. Polymorphism allows programmers to process objects from a variety of different subclasses together efficiently within loops. There are many other advantages of polymorphism which help to reduce the



Polymorphism

The ability to appear in many forms. In OOP this means at runtime a method can process data differently depending on the circumstances.

complexity of code. Because at runtime the system decides which particular method to execute then the programmer does not need to include decisions (often "if" statements) within their code to make such decisions. This results in cleaner and more maintainable code and also faster code execution because the decisions are being made by a built-in part of the system rather than the programmers logic.



GROUP TASK Discussion
Polymorphism is not an easy concept to understand. Create a list of animals that could be viewed as polymorphic. Explain your answers.

As an example of polymorphism consider the execution of the following Java code:

```
1    Character[] myCharacters = new Character[3];

2    myCharacters[0]= new Goodie("Mr Nice",8,2,3,4);
3    myCharacters[1] = new Baddie("Grizzly",5,6);
4    myCharacters[2] = new Character("Crazy Tree",7,8);

5    for(int i=0; i<3; i++) {
6       myCharacters[i].print();
7    }
```

The above code produces the following output:

```
Goodie
My name is Mr Nice
Agility = 8
Health = 2
Personality = 3
Intelligence = 4

Baddie
My name is Grizzly
Agility = 5
Health = 5
Aggresion = 5
BadType = 6

My name is Crazy Tree
Agility = 7
Health = 8
```

In line 1 `Character[] myCharacters` is telling the compiler that the variable `myCharacters` will hold an array of objects of type `Character`. The right hand side code `= new Character[3]` creates an array of `Characters` with 3 elements and initialises the variable `myCharacters` with this array. In Java the 3 elements are indexed beginning with 0.

Line 2, 3 and 4 are instantiating and assigning actual objects to each of the array elements. Note that both `Goodie` and `Baddie` objects are also `Character` objects because both `Goodie` and `Baddie` classes inherit from the `Character` class. It is true to say that a `Goodie` is a `Character` and also that a `Baddie` is a `Character`. Because this is true we are able to store `Goodie`, `Baddie` and also `Character` objects within variables declared to be of type `Character`. This is critical to understanding the concept of polymorphism.

Lines 5, 6 and 7 iterates through the `myCharacters` array executing the `print()` method for each object. This is where the benefits of polymorphism can be seen. If the object is a `Goodie` then the `print()` method from the `Goodie` class executes, if the object is a

Baddie then the Baddie class print() method executes and finally for Character objects the Character.print() method executes. These decisions about the most appropriate method to execute occurs despite the array being initialised to hold Character objects.



GROUP TASK Activity
Work through the code above to confirm the output is indeed correct. Pay particular attention to the decisions made by the runtime system as it decides which print() method should be executed.



GROUP TASK Discussion
Inheritance, abstraction and overriding are critical to the operation of polymorphism. Discuss the connection between inheritance, abstraction, overriding and polymorphism.



HSC style questions:

Question 1.

Consider the following in relation to object-oriented paradigm concepts.

All cars, buses and bicycles are vehicles. All vehicles have travelled a certain distance since new, have a particular number of wheels and seats, use some type of fuel, are able to move and can turn.

However, moving and turning cars and buses is quite a different process to moving and turning a bicycle. As a vehicle moves it travels a certain distance, and the distance travelled since new is stored within the vehicle's odometer. The only way to alter the value in the odometer is to move the vehicle.

Based on the above information:

(a) Describe an example of a class including its data and methods.

(b) Identify an example of inheritance and use your example to explain how inheritance improves the productivity of programmers.

(c) Describe an example of polymorphism.

(d) Identify an example of encapsulation and explain how encapsulation improves the ability to reuse code.

Suggested solutions

(a) A class could be created called Vehicle which would include the data (or attributes) Wheels, Seats, Fuel and Odometer. Methods (or operations) would include Turn and Move.

(b) Three sub-classes (or child classes), say Car, Bus and Bicycle could inherit data and methods from the parent class Vehicle. This reduces the need to write the sub-classes from scratch as the code within the Vehicle class is reproduced within the sub-classes. In addition, once the code has been thoroughly tested it can be relied upon each time it is reused. The Bicycle class would have its own distinct Move and Turn methods which would override those inherited from the parent Vehicle class.

(c) Assume one object from each of the classes Car, Bus and Bicycle has been created (instantiated). As all objects are Vehicles then the methods Move and Turn can be used to manipulate all three. However at runtime the Move method for the Bicycle object performs quite different processing to the Move method of the Car and Bus objects. Similarly for the Turn method. Both the Move and Turn methods are said to be polymorphic as the system chooses the appropriate method to execute at runtime.

(d) Encapsulation means the value of data (attributes) of an object can only be altered by that object's methods. In the example the Odometer attribute can only be altered by the Move method of that object. Encapsulation allows programmers to create robust objects that are suitable for reuse. As data can only be changed by methods the programmer can test and validate all inputs thoroughly within the objects methods before any of the object's data is changed. This means unexpected inputs can always be dealt with appropriately by the object rather than causing errors to occur within the larger program.

Question 2.

The following Java code defines 3 classes:

```
class Animal {
  private String name = "not set";
  private static int numAnimals = 0;
  public Animal() {
    numAnimals++;
  }
  public void setName(String n) {
    if (n.length() != 0) name = n;
  }
  public String getNumAnimals() {
    return (numAnimals + " Animals");
  }
  public String getName() {
    return (name + " is an animal");
  }
}

class Bird extends Animal {
  private static int numBirds = 0;
  public Bird() {
    super();
    numBirds++;
  }
  public String getNumBirds() {
    return (numBirds + " Birds");
  }
  public String getName() {
    return (super.getName() + " and a bird");
  }
}

class Parrot extends Bird {
  private static int numParrots = 0;
  public Parrot() {
    super();
    numParrots++;
  }
  public String getNumParrots() {
    return (numParrots + " Parrots");
  }
  public String getName() {
    return (super.getName() + " and a
parrot");
  }
}
```

The following Java code is executed:

```
class MyAnimalApp {
  public static void main(String[] args) {
    Animal a = new Animal();
    a.setName("Dumbo");
    System.out.println(a.getName() );
    Animal b = new Bird();
    b.setName("Flighty");
    System.out.println(b.getName() );
    Animal p1 = new Parrot();
    p1.setName("Cracker");
    System.out.println(p1.getName() );
    Parrot p2 = new Parrot();
    p2.setName("Polly");
    System.out.println(p2.getName() );
    System.out.println(p2.getNumAnimals() );
    System.out.println(p2.getNumBirds() );
    System.out.println(p2.getNumParrots() );
  }
}
```

The following output is produced when the above code is executed:

```
Dumbo is an animal
Flighty is an animal and a bird
Cracker is an animal and a bird and a parrot
Polly is an animal and a bird and a parrot
4 Animals
3 Birds
2 Parrots
```

(a) Identify an example of each of the following within the above code.

(i) Class

(ii) Instance of an object

(iii) Class variable (or attribute)

(iv) Instance variable (or attribute)

(v) Method

(vi) Instantiation

(b) The code below only occurs within the Animal class definition, however it is executed using objects of type Animal, Bird and Parrot.

```
public void setName(String n) {
    if (n.length() != 0) name = n; }
```

Identify and outline the object oriented concept that makes this possible.

(c) Identify and describe an example of *polymorphism* within the above code.

(d) Identify examples within the code that illustrate the concept of *encapsulation*.

Suggested solutions

(a) *Only one of each of the following is required*

(i) Class: Animal, Bird, Parrott, MyAnimalApp

(ii) Instance of an object: a, b, p1, p2

(iii) Class variable (or attribute): numAnimals, numBirds, numParrots

(iv) Instance variable (or attribute): name

(v) Method: Animal(), setName(String n), getNumAnimals(), Bird(), getNumBirds(), Parrot(), getNumParrots(), getName(), main(String[] args)

(vi) Instantiation: new Animal(), new Bird(), new Parrot()

(b) Inheritance is the OO concept that makes this possible. The super class Animal implements the method setName(String n). The Bird and Parrot classes are subclasses of the Animal super class, as Bird extends Animal and Parrot extends Bird. This means that the subclasses inherit all the attributes and methods of their super class (Animal) which includes the setName (String n) method.

(c) An example of polymorphism is the getName() method found in the subclasses Bird and Parrot. Although this method has the same name (and call signature) in each Class it performs different processing. The getName() method is found in the super class Animal, but the Bird and Parrot subclasses override and redefine this method, to create their own implementation. For example, Bird redefines the getName method to call the super class (Animal) implementation of getName and then also perform additional operations.

In addition Parrot is a Bird and Bird is a Animal, therefore both Bird and Parrot are also Animal objects. This means that a, b and p1 can be declared to be Animals and then assigned objects of either Animal, Bird or Parrot type. When the overridden method getName() is executed the version of the method from the appropriate class is executed. This is an example of polymorphism in action.

(d) Examples illustrating encapsulation are the private attributes in the Animal, Bird, and Parrot classes combined with methods being the only way to alter these attributes.

For example, the numBirds class variable within the Bird class. This variable has the "private" identifier meaning that the variable can only be accessed through a public method within the class. Continuing with this example, this variable can only be altered using the constructor (initialisation method) Bird() and to get the value you need to use the method getNumBirds(). This is designed to protect the attribute from being altered and accessed inappropriately from code outside the class, thus protecting the integrity of the attribute's data.

Similarly the name instance variable present in the Animal class, which is inherited by the Bird and Parrot classes, cannot be altered or accessed directly. One must use the setName() method to alter or the getName() method to access the name instance variable. The setName() method performs some limited validation on the value passed to the method before the actual name variable is altered. This protects the attribute from inappropriate changes.

Notes

- This question includes class variables and instance variables which are both types of attributes. A class variable exists once for each particular class therefore when an object changes a class variable the changed value is reflected across all objects of that class. Instance variables exist within each individual object. That is each object has its own set of instance variables which are separate to other object's instance variables. If an object changes the value of one its instance variables this has no effect on the value of any other object's variables.



GROUP TASK Activity
Perform a desk check of the code in question 2 to ensure you are able to reproduce the output provided in the question.



GROUP TASK Activity
Enter the Java code from the question into a Java editor. Create further classes for Finches, Mammals, Monotremes and Marsupials which include similar attributes and methods as the existing classes. Create an array of at least 10 animals of all different types and efficiently print out the data returned by each animal's getName() method.

SET 9C

1. What is the term used to describe programs that react to particular user inputs?
   (A) Object oriented.
   (B) Event driven.
   (C) Procedural.
   (D) Functional.
2. An object's data is hidden and can only be altered via one of its public methods. This is a description of
   (A) encapsulation.
   (B) abstraction.
   (C) inheritance.
   (D) polymorphism.
3. Abstraction includes the process of
   (A) hiding an object's data and processes from it's environment.
   (B) coding a précis or summary of the objects.
   (C) selecting the common features of objects and procedures.
   (D) taking on the attributes of another object.
4. In terms of OO programming, what provides the interface that allows objects to communicate with other objects?
   (A) The object's attributes.
   (B) The object's data items.
   (C) The object's class.
   (D) The object's methods.
5. The term Class can best be described as
   (A) a category of objects.
   (B) a category of attributes.
   (C) a category of methods.
   (D) a category of processes.
6. Inheritance is the ability of objects to take on the characteristics of
   (A) child classes.
   (B) sub classes.
   (C) parent classes.
   (D) Both (A) and (B).
7. A constructor is used to
   (A) initialise an object during its creation.
   (B) assign the object to its applicable class.
   (C) create the object's class definition.
   (D) None of the above.
8. Overriding can be described as
   (A) a method with the same name but a different number of parameters as another method.
   (B) the ability to inherit the attributes of another class.
   (C) A subclass method with identical name and parameters as its parent class.
   (D) the ability of an object to create a second mirror-image copy of itself.
9. Creating an object based on a class is called
   (A) instantiation.
   (B) abstraction.
   (C) polymorphism.
   (D) encapsulation.
10. Which of the following can be described as an object oriented language?
    (A) Java.
    (B) C++.
    (C) Smalltalk.
    (D) All of the above.

11. Objects contain both attributes and methods. It is not possible to alter an object's attributes directly from another object. What term is used to describe this concept? Wouldn't it be better if attributes could be altered directly? Discuss.

12. A frog progresses through three stages. First it is an egg, then a tadpole and then finally a frog. Although all forms are the same species and are genetically identical they appear and behave quite differently. Which object oriented concept could be used to describe this scenario? Give an example of the use of this term as it is used in OOP.



13. Inheritance, encapsulation and abstraction are all vital aspects of all object oriented languages. Explain the meaning of each of these terms. How do these concepts improve the reusability and maintainability of code?

Use the following scenario to answer questions 14 and 15.

A program is being developed using an object oriented programming language. This program is designed for use by pre-school children. A simulated adventure playground is the backdrop for the game. The aim of the game is to try to please both the children and the animals within the playground. The pre-schooler does this by moving children and animals to more favourable positions within the playground. The interaction of items provides the incentive for children to rearrange items into more appropriate positions. There are three different types of item within the playground, animals, play equipment and children. Individual items can interact both with the user and with other items. For example, a child may play on a piece of play equipment or a dog could look at the user and bark. Children and animals are able to move freely around the playground, play equipment cannot.



14. Develop a hierarchical system of classes that could be used within this project. Describe the attributes and methods you envisage may be required within each class.

15. Create a set of at least six objects that might exist within the playground. Write down the initial attributes of each of your items. What will you use to determine the result of interactions with other objects? Explain your answer using examples.

PROGRAMMER PRODUCTIVITY

The overriding motivation for introducing programming languages based on different paradigms is to increase the productivity of programmers. Different problems require different sets of tools to enable the production of efficient and reliable solutions. The effect of using languages more suited to particular tasks has an effect on productivity. Some of these affects will improve productivity and others may reduce productivity. Management must weigh up the positive and negative effects when deciding on a choice of language to be used for particular projects. Some areas influencing productivity include:

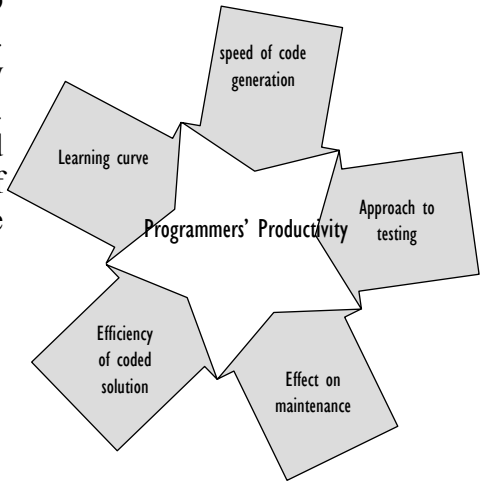


Fig 9.17
New languages affect programmers' productivity in various ways.

- speed of code generation
- approach to testing
- effect on maintenance
- efficiency of solution once coded
- learning curve (training required)

Let us consider these areas in more detail.

Speed of code generation

The speed at which code can be generated is a traditional method of measuring a programmer's productivity. Many software companies still pay programmers based on the number of lines of code they write. Thankfully, this method of payment has been largely phased out; encouraging programmers to write long-winded source code is certainly not the way to develop efficient and elegant programs. Nevertheless, the speed at which a programmer can create code to solve a problem is a valuable measure of productivity.

Programming languages that increase the speed of code generation must increase the productivity of programmers. Similarly, using a language based on a more suitable paradigm will also increase the speed of code generation. Different problems are suited to solutions using different paradigms. Choosing the most suitable paradigm can make the process of software design and code generation more efficient and will result in a more elegant and useable final solution. Let us examine an example to illustrate:



Consider the following:

A software development company is working on a new computer game called *Teenager's Revenge*. This game has various characters that interact with each other; each character is either a teenager or an adult. The teenagers possess qualities in common and similarly the adults have qualities in common. Teenagers are able to argue with adults and adults are able to attempt to reason logically with teenagers. Adults are split into sub-types; parents, relatives, teachers and friends. Of course, the teachers are the most understanding and logical, whereas the parents are the least logical but the most responsible from the teenager's viewpoint.

This product suits development using an object oriented paradigm. Each character within the game would be an object with data and methods. For example, a parent would be an adult object containing low levels of logic combined with a high level of responsibility and a medium level of understanding. Each parent object would have a method attached to explain how they reason with teenagers of different types. Teachers would be adult objects with high levels of understanding and logic and medium levels of understanding. They too would have methods attached to describe how they interact with teenagers. Similar objects could be created for each of the teenage characters.



GROUP TASK Discussion
Could the imperative or logical paradigm be used to develop the above product? Why would these paradigms be less suitable than the object oriented paradigm?

Approach to testing

In chapter 6, we examined the testing phase of the software development cycle. Much of what we learnt is relevant to testing software produced using languages based on any of the paradigms.

Unit testing, or testing individual modules, is perhaps where the largest productivity gains can be made. This is particularly the case when using object oriented languages. These languages force programmers to write self-contained classes. These classes (or the objects produced) can be thoroughly tested without requiring reference to other components. Imperative languages do not force programmers to develop such self-contained units. Classes that have been thoroughly tested can be treated as black boxes; the internal processing details need not be retested.

Once a class has been created it is possible to create new classes that inherit the attributes and methods from the original. This ability means that much of the code produced within new software projects will be tried and tested code from previous efforts. The previous section on OOP explained these concepts in more detail. As a consequence, testing is simplified resulting in productivity increases.

It is difficult to discuss testing without mentioning test data. As we said in chapter 6, the creation of test data sets is a vital stage of the testing process. Languages that can encapsulate their data in such a way that it cannot be altered by other processes will greatly reduce the magnitude and breadth of the final test data sets. OOP languages discussed in the previous section not only encourage data encapsulation, in some cases they enforce it.



GROUP TASK Discussion
Explain how encapsulation and inheritance can assist in the process of testing the source code of a newly developed software product.



GROUP TASK Discussion
Testing should be an ongoing process throughout the development of a software product. In what ways do each of the new paradigms assist in the ongoing testing process? Discuss.

Effect on maintenance

Maintenance is primarily about the ability of the code to be modified to meet changed requirements. To modify code requires locating the section of code that requires modification and then actually implementing the changes within the code. This requires thorough documentation throughout the software development cycle but particularly at the design and implementation stages.

Isolating the precise location of the code to be altered can be a time consuming and difficult task, particularly for large applications. Languages that force programmers to modularise their work greatly assist maintenance programmers to locate particular procedures and functions within the source code. Object oriented languages do this well, imperative and logical languages certainly provide the facility but it is not enforced. Ultimately no matter what language is used, programmers can write well-structured, elegant and readable code and they can also write rubbish.

Large software companies are reluctant to develop products using languages that are not widely understood. This is a reasonable and responsible reaction given the limited number of programmers available to maintain the code. If a large bank decided to convert all its applications to a new language then they must ensure maintenance personnel are available to maintain the product.

Robust, bug-free code is the desire of all software companies for it minimises maintenance issues. Many of the concepts we have covered in this chapter encourage or enforce the development of such code. Logical languages such as Prolog operate at a higher conceptual level thus removing the programmer from the technical details at the CPU level. The responsibility for managing memory and CPU operations is removed from the programmer and handed to the language and it's developers.



GROUP TASK Discussion

In terms of fault maintenance, what are the advantages and disadvantages of using higher-level languages? Justify your answers.

Efficiency of solution once coded

Efficiency of software once it is coded is often measured in terms of speed. How fast can an application retrieve a record, process it and store the result? How many CPU cycles does it take for a word processor to prepare a ten-page document for printing? These types of tests are designed to test the efficiency of the final solution.

Imperative languages are based on the von Neumann architecture. They have evolved in line with the developments in hardware technologies. Many of these languages e.g. C enable the programmer to work closely with the CPU's operations. Well written C programs will out-perform most similar programs written using OOP or logical programming languages. Unfortunately, this is a consequence of the development and evolution of hardware technologies rather than software technologies. Essentially, languages using other non-imperative paradigms are, in a sense, crippled by the hardware. They are not able to compete on an even playing field.

The good news is that these languages can considerably reduce development times. In addition, hardware, although not designed for these paradigms, is now capable of executing applications at such a speed that efficiency concerns are often of reduced importance. This is particularly true of many popular OOP languages such as Java and C++.



GROUP TASK Discussion
Explain the reasons why applications developed with Prolog are often unable to operate with the speed of similar products produced with more traditional languages.



GROUP TASK Research
Java and also .NET languages use an intermediate code which then runs on a virtual machine. Research to determine advantages and disadvantages of intermediate code such as Java's bytecode and .NET's CIL code.

Learning curve (training required)

Programming languages based on the logical programming paradigm (and other paradigms such as the functional paradigm no longer studied in SDD) have not gained wide acceptance amongst the general software development community. They are often viewed as obtuse and specialised. Many programmers view them as interesting oddities and hence they are only used in specialised areas. It can be argued that intensive and ongoing training and marketing is required to lift the status of these languages. It is a difficult task to thoroughly learn languages based on new paradigms and often the learning curve is steep. The benefits however are often greater than first envisaged.

Object oriented languages, on the other hand, are accepted and used by a large proportion of software development companies. Most universities now teach object oriented languages as part of their computing and engineering degrees. It is true that the languages we first learn are the ones with which we feel most comfortable. Because many programmers are experiencing and learning OOP techniques early in their careers these languages have gained wide acceptance.

Try searching the internet for tutorials on particular imperative, logical, object oriented and functional languages. You'll most likely find hundreds on C, Basic, Pascal, Fortran and various other imperative languages. You'll find an almost equal quantity on C++, Java and other OOP languages. What about Prolog, or functional languages such as APL, Haskell or Lisp? Your search will of course reveal them, but specialist interest groups will have written most with not many available as commercial material.

It is not an easy task to learn any new programming language, but when the paradigm on which it is based is also new the problem is compounded. There is no doubt that software developers and the greater community will benefit if they can find the time to examine and learn new ways of doing things.



GROUP TASK Discussion
Imagine you are a member of a logical programming language's international interest group. What recommendations would you make to this group in regard to training new users? Discuss.



GROUP TASK Discussion
Many of the concepts introduced in this chapter are new and require concentration and perseverance to master. Share your experiences with the class. Use the classes combined input to develop a set of training recommendations for future students doing this topic.



HSC style questions:

Question 1.

Imperative languages use control structures of sequence, selection and repetition together with local and global variables. Explain how logic languages such as Prolog achieve the same outcomes without all of the control structures, local and global variables.

Question 2.

(a) Describe ONE significant reason for the development of the *object oriented* paradigm.

(b) Explain the use of *encapsulation* within the *object oriented* paradigm and provide an example.

(c) Discuss features of the *object oriented* paradigm that affect the programmers' productivity with particular reference to speed of code generation.

Question 3.

A scientist is trying to classify various rock materials. Many experiments have been performed to determine the features and characteristics of each sample. Features and characteristics include colour, atomic weight and structure, abundant elements plus location found.

With all of this information the scientist has discovered that it possible to classify any rock material into one of the categories.

The scientist wants his assistants to be able to carry out the classification process without him present and therefore requires a system to be developed which accepts the necessary information about the sample and classifies it.

Recommend the most appropriate paradigm for the solution of this problem. Explain why other paradigms would be less appropriate.

Suggested solutions

Question 1.

Imperative paradigms use the repetition control structure to repeat statements whereas the logical paradigm uses recursion. The predicate is called initially and for each subsequent call the argument is modified until it reaches the termination condition. Logical languages have a stopping (or terminating) condition in their code. Selection in Prolog is implemented by having different versions of each predicate which will match with different facts or rules.

With regards to variables, imperative languages allow the use of global variables resulting in a situation where a value can be seen and hence modified anywhere within the program. With logical languages values can only be seen in a predicate if they are passed as parameters. Prolog variables are used to match or bind with known facts during the goal seeking process. This is different to imperative language variables which are assigned values specifically by the programmer or by the detailed logic of the solution.

Question 2.

(a) One significant reason for the development of OOP was that it simplifies the reuse of code. This reduces repetitive programming tasks and is achieved through the use of classes of objects and their associated data and methods. Once a class has been defined it can be reused in any programming project without the need to redefine the data structures and operations.

(b) The principle of encapsulation is essential to the OOP. It establishes a firewall between the user of an object and the code implementing it, thus achieving information hiding. In particular the user does not need to know how the data or attributes of an object are defined. If we had a Stack class with pushing and popping operations the stack could be implemented as an array, a list or whatever, since the representation of the Stack is unknown outside of the object. The only way to interface with the object is via the methods.

(c) OOP increases programmers' productivity because of the inheritance feature. Once a class has been defined and tested it can be used by the programmer. If the programmer wishes to extend the class into a sub class they do not need to rewrite, and hence test, all of the methods needed to operate an object of the sub class but can call the methods from its super or parent class, this means only additional methods need to be defined and tested which increases code generation time.

Question 3.

The logic paradigm would be the most appropriate as the scientist has come up with some rules on how to classify the rock material. These rules as well as all the known facts about existing rocks can be entered into a Prolog system or perhaps into an expert system shell. The assistant will then be able to enter the facts about the current sample to be classified and the logic paradigm (or expert system) will then use its rules to determine the correct classification.

The imperative and OOP paradigms are unsuitable. Both these paradigms require algorithms and then code which explicitly describes how to solve the classification problem. A different algorithm (and hence subprogram) would be needed to solve each different classification method. Therefore, any OOP or imperative program written to solve this problem will be long and complicated with lots of cases listed. Subsequently, it can be seen that the most appropriate paradigm is logic.

Notes

- Questions which ask about the most appropriate paradigm do not necessarily have a single correct answer. It is often possible to describe a suitable method of implementing a solution using various different paradigms.
- The logic and object oriented paradigms can be used together as the concepts involved are very different and meet very different purposes. There are object oriented versions of Prolog which attempt to combine the advantages of both logic and object oriented programming.

CHAPTER 9 REVIEW

1. Some areas that have an influence on productivity include
   (A) effect on maintenance.
   (B) approach to testing.
   (C) speed of code generation.
   (D) all of the above.

2. If you as a programmer were considering variables and control structures for the design of your computer program, which paradigm would you have most likely chosen to work with?
   (A) Imperative.
   (B) Logic.
   (C) Object oriented.
   (D) Functional.

3. A goal in Prolog terms, can best be described as
   (A) a fact contained in the knowledge database.
   (B) a rule contained in the knowledge database.
   (C) a query that results in either being fulfilled or not being fulfilled.
   (D) a query that results in being fulfilled.

4. You are writing a computer program and have created a class that contains multiple methods with exactly the same name. This does not concern you because you are coding in
   (A) an object oriented language.
   (B) an imperative language.
   (C) a logical language.
   (D) any of the above.

5. Which term refers to the ability to run code in response to a user's action?
   (A) Imperative.
   (B) Event driven.
   (C) Object oriented.
   (D) Logic.

6. You have been employed by a company that specialises in the development of artificial intelligence systems. With which paradigm will you most likely have to become proficient?
   (A) Relational.
   (B) Imperative.
   (C) Object oriented.
   (D) Logic.

7. With regards to software programs, maintenance can be described as
   (A) the ability of the code to be modified to meet changed requirements.
   (B) the development of robust code.
   (C) modularised coding.
   (D) a language that encapsulates data.

8. Which type of languages are based on the von Neumann architecture?
   (A) Logic.
   (B) Imperative.
   (C) Event driven.
   (D) Object oriented.

9. Backward and forward chaining can best be described as
   (A) two common strategies used to resolve goals.
   (B) two facts contained in the knowledge base used to resolve goals.
   (C) two rules contained in the knowledge base used to resolve goals.
   (D) none of the above.

10. The process of designing objects by breaking them down into component classes is the process called
    (A) encapsulation.
    (B) inheritance.
    (C) polymorphism.
    (D) abstraction.

11. In this topic, we have studied three programming paradigms; imperative, logical and object oriented. List and describe the main components or building blocks used in each of these paradigms.

12. The choice of programming paradigm can have a significant effect on programmers' productivity. In fact the main reason for choosing a particular paradigm is often to increase programmers' productivity. List and discuss five effects on programmers' productivity.

Consider the following problem when answering questions 13, 14 and 15.



Snakes and ladders is a common children's board game. Each player takes it in turns to throw a pair of dice. The player then moves his or her piece the number of squares indicated on the dice. If they land on a square containing the bottom of a ladder then their piece is moved to the square at the top of the ladder. If a player's piece lands on the top or head of a snake then their piece moves down to the square indicated by the end of the snake's tail. The first player to reach the last square on the board wins.



Imagine you are considering creating a computer-based simulation of this game.

13. Explain how the game is played from an imperative paradigm point of view. Would this be a suitable paradigm in which to develop this game?

14. Explain how the game is played from a logical paradigm point of view. Would this be a suitable paradigm in which to develop this game?



15. Explain how the game is played from an object oriented paradigm point of view. Would this be a suitable paradigm in which to develop this game?



In this chapter you will learn to:

- effectively use an ASCII table to convert a character to its equivalent ASCII value and viceversa
- recognise the relationship between upper and lower case letters and digits, and their ASCII representation
- use the Unicode table which represents a larger character set than is available with ASCII
- convert a binary or hexadecimal representation to its equivalent character from the ASCII or Unicode table
- represent a string of binary digits as its hexadecimal equivalent and vice versa
- convert integers between binary, decimal and hexadecimal representations
- convert between decimal fractions and the equivalent IEEE754 single precision floating point representation
- recognise implications of the limitations of particular data representations
- perform arithmetic operations in binary
- generate truth tables for a given circuit
- describe the function of a circuit from its truth table
- design a circuit to solve a given problem
- convert between the Boolean representation of a circuit and its circuit diagram
- build and test both user-designed and specialty circuits using integrated circuits or simulation software
- use a cyclical approach when designing circuits
- modify an existing circuit design to reflect changed requirements
- describe the function of specialty circuits
- analyse a specialty circuit in order to determine its output
- explain how a flip-flop can be used in the storage and shifting of a bit in memory
- interpret a data stream for a device for which specifications are provided
- modify a stream of data to meet changed requirements, given the hardware specifications
- generate a data stream to specify particular operations for a hardware device, for which specifications are provided such as a printer, to specify line feed, form feed, font and style change, and line spacing
- develop an algorithm to identify and extract data and/or control characters in order to interpret a data stream sent from the hardware
- develop an algorithm to generate a data stream to provide relevant instructions to the hardware
- recognise that a string of binary digits can have many different meanings
- interpret a string of binary digits, given a number of different possible specifications

Which will make you more able to:

- explain the interrelationship between hardware and software
- describe how the major components of a computer system store and manipulate data
- explain the interrelationship between emerging technologies and software development
- identify needs to which software solutions are appropriate
- create and justify the need for the various types of documentation required for a software solution
- select and apply appropriate software to facilitate the design and development of software solutions

In this chapter you will learn about

Representation of data within the computer

- character representation, namely:
  - ASCII
  - Unicode
- representation of data using different number systems
  - binary
  - hexadecimal
  - decimal
- integer representation, including:
  - sign and modulus
  - 1's complement
  - 2's complement
- floating point/real representation
  - very large positive and negative values
  - very small positive and negative values
  - integer and non-integer values
  - limitations
- binary arithmetic, including:
  - addition
  - subtraction using 2's complement representation
  - multiplication (shift and add)
  - division (shift and subtract)

Electronic circuits to perform standard software operations

- logic gates, including:
  - AND, OR, NOT, NAND, NOR, XOR
- truth tables
- Boolean algebra
  - describing a circuit
  - simplifying an existing circuit
- circuit design steps
  - identify inputs and outputs
  - identify required components
  - check solution with a truth table
  - evaluate the circuit design
- specialty circuits, including:
  - half adder, full adder
  - flip-flops

Programming of hardware devices

- the data stream
  - format of the data stream
    - header information, data block, trailer information
  - use of control characters
  - use of hardware specifications to describe the expected format of the data stream
- processing an input data stream from sensors and other devices
  - the structure of the data stream
    - the need to recognise and strip control characters
    - the need to identify the data characters
  - interpreting the data stream
  - devices such as: security cameras, USB mouse, biometric scanners
- generating output to an appropriate device
  - determining the purpose of the output
  - the structure of the data stream
    - required header information
    - the need for control characters
    - specification of data characters
    - required trailer information
  - devices such as: model helicopter, model train/car/boat, plotter, cutter machines, automated jar filling line with valves and conveyor belt, scrolling display system for one line text display, modem, printer
- issues with interpreting data streams

10

OPTION 2
THE INTERRELATIONSHIP BETWEEN SOFTWARE AND HARDWARE

In this option, we examine the basis of digital computer hardware; data, storage and processing. We consider the manner in which data is represented within the computer. Different data types require different methods of binary representation; all are based on the binary number system. We then examine the circuits that allow us to store and process this data. Logic gates are the building blocks for integrated circuits. What are logic gates and how are they combined to perform new and varied functions? We answer these questions and design some simple integrated circuits. In particular, we investigate circuits used to perform binary addition and those used to store binary data.

The final section of this option describes the programming and interfacing of hardware devices. We examine the data streams required to provide input to various hardware devices as well as the returned output data streams. The format and processing required to generate and understand these data streams, is discussed. We look at a number of practical examples where this information is utilised to control hardware devices in real world applications.



Consider the following:

Computers can only process binary data; that is strings of zeros and ones. Considering this fact, it seems remarkable that computers are able to perform such a broad range of tasks. How can strings of binary digits be used to show videos, track bank account transactions or control a production line of robots? Many people consider the computer to be the most significant invention of all time.



GROUP TASK Discussion
Develop a list of tasks that would be impossible to perform without the assistance of computers. How are computers able to perform all these tasks using just binary numbers and combinations of circuits?



GROUP TASK Discussion
Why would many people consider the computer to be the most significant invention of all time? Consider prior inventions and discoveries that were necessary before the computer, as we know it could be developed.

REPRESENTATION OF DATA WITHIN THE COMPUTER

All data within a computer system is stored and processed in binary. The binary or base 2 number system, uses only 2 digits; zero and one. These binary digits or bits can be stored in a variety of ways depending on the particular medium. For example, lands and pits on a CD-ROM, magnetised needles on the surface of a floppy disk or high and low voltages sent down a communication path. Later in this chapter, we will examine how binary is stored and processed using integrated circuits.



Fig 10.1
Hard disk drives, CDs, DVDs and flash memory all store binary data.

Before considering methods of representing data using binary numbers, we need an understanding of how the binary system operates. Once we have an understanding of the binary system, we shall consider common methods for representing characters, integers and fractions. We conclude by examining binary arithmetic; addition, subtraction, multiplication and division.

THE BINARY NUMBER SYSTEM

The decimal number system is based on the number ten hence we require ten symbols. We call these symbols digits; in decimal or base 10 we use the digits 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9. The binary number system is based on the number two, hence we require 2 symbols or digits, and we use 0 and 1.

The placement of each digit in relation to other digits determines its place value. For example, the decimal number 3407 means three thousands, four hundreds, zero tens and seven units. The four is said to have a place value of 100, or 10^2. The three has a place value of 1000 or 10^3. In other words, 3407 could be written as $(3 \times 10^3) + (4 \times 10^2) + (0 \times 10^1) + (7 \times 10^0)$.

Decimal Number	10^3=1000	10^2=100	10^1=10	10^0=1
3407	3	4	0	7
968	0	9	6	8
1082	1	0	8	2
6600	6	6	0	0
123	0	1	2	3
0	0	0	0	0
9999	9	9	9	9

Fig 10.2
Decimal numbers have place values based on powers of the number ten.



GROUP TASK Activity
Write down a series of ten decimal numbers between 0 and 1 million. Create a table, such as the one in *Fig 10.2*, and complete the table using your ten numbers.

Binary numbers are based on the number two thus place value is determined by powers of two. The binary number 1101 can be written as $(1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0)$. In decimal, the binary number 1101 has a value of 8 + 4 + 0 + 1, which is equal to 13. We write this as $1101_2 = 13_{10}$, the subscript is used to indicate the base used. Leading zeros have no effect on the value of a number. For example, 00010111_2 is equivalent to 10111_2. Zeros between non-zero digits are required to ensure correct place value. For example, 1111_2 is not the same number as 101110_2.

When using binary numbers with computer systems it is usual to retain the leading zeros to make up numbers with a multiple of 8 bits. For example, we write 00111011_2 rather than 111011_2. This is because computers deal with binary numbers of a specific length. Generally, binary numbers are expressed in series of 8 bits, called a byte. Most computers store and process data in multiples of 8 bits; the number of bits processed at a time is called the word length of the processor. Common processors have word lengths of 8, 16, 32 or 64 bits.

Binary Number	2^7=128	2^6=64	2^5=32	2^4=16	2^3=8	2^2=4	2^1=2	2^0=1
11010111	1	1	0	1	0	1	1	1
00101111	0	0	1	0	1	1	1	1
01110111	0	1	1	1	0	1	1	1
01011100	0	1	0	1	1	1	0	0

Fig 10.3
Binary numbers have place values based on the number two.



GROUP TASK Activity
Write down a series of ten binary numbers between 0 and 11111111. Create a table, such as the one in *Fig 10.3*, and complete the table using your ten numbers. What is the largest number you can store using 8 bits?

Counting in binary

As is the case in the decimal system, once we run out of digits we increment the number held to the left and set the current value back to zero. For example, in decimal we count 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12…19, 20, 21…29, 30, 31, etc. In binary, we only have only two digits so our numbers very quickly become long. For example, in binary we count 1, 10, 11, 100, 101, 110, 111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111, 10000, we are already using five digit numbers yet in decimal we have only reached a value of 16.

Fortunately, computers are particularly good at storing vast amounts of data, so the length of binary numbers is relatively insignificant in computing terms. Humans, on the other hand, find binary numbers difficult to accurately read and process. To simplify binary numbers for humans, we use the hexadecimal number system.

Decimal	Binary
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111
16	10000
17	10001
18	10010
19	10011
20	10100

Decimal	Binary
21	10101
22	10110
23	10111
24	11000
25	11001
26	11010
27	11011
28	11100
29	11101
30	11110
31	11111
32	100000
33	100001
34	100010
35	100011
36	100100
37	100101
38	100110
39	100111
40	101000

Fig 10.4
Counting in binary and decimal.



GROUP TASK Activity
The large number of digits used to represent numbers using binary provides difficulties for us mere mortals. For example $1_2=1_{10}$, $10_2=2_{10}$, $100_2=4_{10}$, $1000_2=8_{10}$, $10000_2=16_{10}$, $100000_2=32_{10}$. Extend this sequence to 32 bits, that is a one followed by thirty one zeros.

Hexadecimal numbers

Hexadecimal numbers use the base sixteen number system. Hex meaning six and dec meaning ten so hexadecimal refers to sixteen. Hexadecimal numbers are used because they greatly simplify the task of writing binary numbers yet conversion between base 2 and base 16 is simple. An eight digit binary number, one byte, can be written as a two digit hexadecimal number. Each sequence of four bits, called a nibble, converts neatly into a single hexadecimal digit.

In base 16, we require sixteen unique symbols to use as our digits. Because hexadecimal is so often used on computers, it makes sense to use symbols available on a standard keyboard. We use the sixteen symbols 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E and F as our digits. Furthermore, special symbols are often used to indicate the number following is in base 16. &H13 and 0x13 are the equivalent of 13_{16}, similarly &H6E and 0x6E are equivalent to writing $6E_{16}$.

Decimal	Binary	Hexadecimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Fig 10.5
Decimal to Binary to hexadecimal conversion.

As is the case in other bases, place value is determined by the position of each digit. The magnitude of each place value is calculated using powers of the base number. For example $3E7_{16}$ can be written as $(3 \times 16^2) + (E \times 16^1) + (7 \times 16^0)$ and $50AB_{16}$ as $(5 \times 16^3) + (0 \times 16^2) + (A \times 16^1) + (B \times 16^0)$.



GROUP TASK Activity
Although not strictly required in this course, base 8 or octal numbers are often used in conjunction with computer systems. Make up a table of octal numbers compared with their binary equivalents. Why do you think octal numbers would be of use in regard to computers?

Converting between hexadecimal, binary and decimal

Most electronic calculators include functions to perform conversions between numbers in different bases. In this course, we need to understand the processes taking place to perform these conversions.

- Binary to decimal conversion

To convert a binary number to decimal, we write the binary number in expanded form and then calculate the result. For example, let us convert 110111_2 into decimal.

$$\begin{aligned} 110111_2 &= (1 \times 2^5) + (1 \times 2^4) + (0 \times 2^3) + (1 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) \\ &= 32 + 16 + 0 + 4 + 2 + 1 \\ &= 55_{10} \end{aligned}$$

- Hexadecimal to decimal conversion

Again, we write the number in expanded form and calculate the result. For example, let us convert $5C3_{16}$ into decimal.

$$\begin{aligned} 5C3_{16} &= (5 \times 16^2) + (C \times 16^1) + (3 \times 16^0) \\ &= (5 \times 256) + (12 \times 16) + (3 \times 1) \\ &= 1280 + 192 + 3 \\ &= 1475_{10} \end{aligned}$$

- Decimal to binary conversion

This process is slightly more complex as we have not grown up learning powers of two. There are two common methods of performing these conversions; the first involves subtraction, the second repeated division.

Suppose we wish to find the binary equivalent of the decimal number 109.

Method 1.

1. Firstly, we determine the value of the power of two closest to but smaller than the number. In this case 64 is the value that meets this criteria, as 128 is larger than 109.
2. Subtract this number from the original number. That is 109 – 64 = 45, note that 45 is less than 64, the result will always be less than the number subtracted. The first or left hand digit in the answer is a 1.
3. We now consider half of 64, namely 32. 45 – 32 = 13, the second digit in the answer is also a 1.
4. We now consider half of 32, namely 16. As 16 is greater than 13 the third digit in the answer is a 0.
5. We now consider half of 16, namely 8. 13 – 8 = 5, the fourth digit in the answer is a 1.
6. We now consider half of 8, namely 4. 5 – 4 = 1, the fifth digit in the answer is a 1.
7. We now consider half of 4, namely 2. As 2 is greater than 1 the sixth digit is a 0.
8. Finally, we consider half of 2, namely 1. 1 – 1 = 0, the seventh digit is a one.

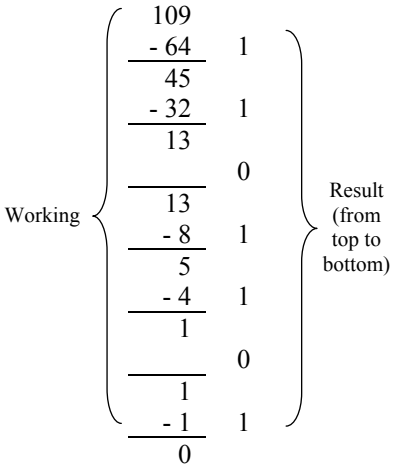


Fig 10.6
Setting out for working using method 1.

Our answer is 1101101_2, so $109_{10} = 1101101_2$. *Fig 10.6* shows the working out required to use this method.

Method 2.

Using this method, we repeatedly divide the number by two. The remainders after these divisions are the answer in reverse order.

1. 109 ÷ 2 = 54 with a remainder of 1. The first or right hand digit is a 1.
2. 54 ÷ 2 = 27 remainder 0. The second digit is a 0.
3. 27 ÷ 2 = 13 remainder 1. The third digit is a 1.
4. 13 ÷ 2 = 6 remainder 1. The fourth digit is a 1.
5. 6 ÷ 2 = 3 remainder 0. The fifth digit is a 0.
6. 3 ÷ 2 = 1 remainder 1. The sixth digit is a 1.
7. 1 ÷ 2 = 0 remainder 1. The seventh digit is a 1.

Again our answer is 1101101_2, so $109_{10} = 1101101_2$. *Fig 10.7* shows the working out required to use this method.

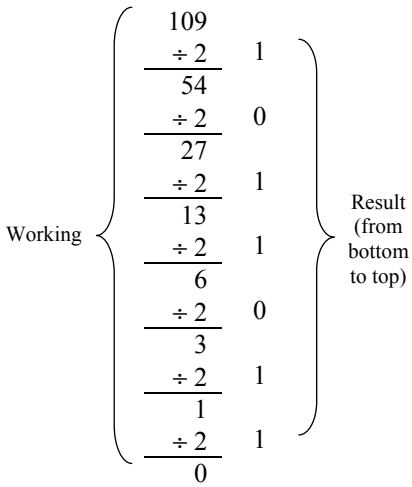


Fig 10.7
Setting out for working using method 2.



GROUP TASK Activity

Use each of the above two methods to convert a series of ten decimal numbers into binary. Use numbers in the range 0 to 255. Why do you think a range of 0 to 255 has been stipulated?

- Decimal to hexadecimal conversion

The essence of the two methods discussed above for converting decimal to binary, can be used for converting decimal to hexadecimal. The difference being that we are working in base 16 rather than base 2. As a consequence, we are working with powers of 16, namely 16^4=65536, 16^3=4096, 16^2=256, 16^1=16 and 16^0=1.

Suppose we wish to find the hexadecimal equivalent of the decimal number 10952.

Method 1

This method requires division by powers of 16, which makes the arithmetic somewhat difficult.

1. Firstly, we determine the value of the power of sixteen closest to but smaller than the number. In this case 16^3 or 4096 is the value that meets this criteria, as 65536 is larger than 10952.
2. Divide this number into the original number. That is 10952 ÷ 4096 = 2 with a remainder of 2760. Note that 10952 – (2 × 4096) = 10952 – 8192 = 2760. The first or left hand digit in the answer is a 2.
3. We now divide 2760 by 16^2 or 256. That is, 2760 ÷ 256 = 10 with a remainder of 200. The second digit in the answer is a 10, which is represented in hexadecimal by the symbol A.
4. We now divide 200 by 16. 200 ÷ 16 = 12 with a remainder of 8. The third digit in the answer is a 12, which is represented in hexadecimal by the symbol C.
5. The final digit must be an 8.

Working	10952		Result (from top to bottom)
	- 8192	2	
	2760		
	- 2560	10=A	
	200		
	- 192	12=C	
	8		
	- 8	8	
	0		

Fig 10.8
Setting out for working using method 1.

Our answer is $2AC8_{16}$, so $10952_{10} = 2AC8_{16}$. *Fig 10.8* shows the working out required to use this method.

Method 2

This method requires repeated division by 16. The remainders forming the result in reverse order.

1. 10952 ÷ 16 = 684 remainder 8. The first or right hand digit in the answer is an 8.
2. 684 ÷ 16 = 42 remainder 12. The second digit in the answer is 12 which is represented by the symbol C.
3. 42 ÷ 16 = 2 remainder 10. The third digit in the answer is 10 which is represented by the symbol A.
4. As 2 is less than 16 this becomes the final digit.

Again our answer is $2AC8_{16}$, so $10952_{10} = 2AC8_{16}$. *Fig 10.9* shows the working required to use this method.

Working	10952		Result (from bottom to top)
	÷ 16	8	
	684		
	÷ 16	12=C	
	42		
	÷ 16	10=A	
	2		
	÷ 16	2	
	0		

Fig 10.9
Setting out for working using method 2.



GROUP TASK Activity

Use each of the above two methods to convert a series of ten decimal numbers into hexadecimal. Use numbers in the range 0 to 65535.

- Binary to hexadecimal and hexadecimal to binary conversion

Converting between base 2 and base 16 is relatively simple as 16 is a power of 2. Each group of four bits, called a nibble, converts neatly into a single hexadecimal digit. Similarly, each hexadecimal digit converts neatly into a nibble of 4 bits. *Fig 10.5* shows the conversion between each hexadecimal digit and binary, it is worthwhile committing this table to memory.

Suppose we wish to convert the binary number 1011010110 into hexadecimal.

1. Firstly, we split the number into nibbles commencing with the right hand side and adding leading zeros if necessary. That is, 1011010110 becomes 0010 1101 0110.
2. Convert each nibble into its hexadecimal digit equivalent. $0010_2 = 2_{16}$, $1101_2 = D_{16}$ and $0110_2 = 6_{16}$.
3. These hexadecimal digits become the answer. Thus $1011010110_2 = 2D6_{16}$ which can also be written as &H2D6 or 0x2D6.

Suppose we wish to convert $A30_{16}$ into binary.

1. A means 10. This is represented by the binary nibble 1010.
2. 3 is represented by the binary nibble 0010.
3. 0 is represented by the binary nibble 0000.
4. Combining these nibbles gives the result 101000100000_2.



GROUP TASK Activity
Create a series of 10 two and three digit hexadecimal numbers. Convert each of these numbers into binary and then reverse the process to convert them back into hexadecimal.

CHARACTER REPRESENTATION

Characters must be represented using binary code if they are to be stored and processed by computers. There are two types of characters that require representation, printable characters and non-printable characters. The printable characters include all the upper and lower case letters of the English alphabet together with numbers, punctuation marks and some other special characters such as the percent and dollar sign. In addition there are thousands of characters which form part of the numerous languages throughout the world. Finally there are non-printable characters which are often called control characters. These characters are not displayed on the screen but rather are used to perform commonly required functions. For example, backspace and delete are generally used to remove data, carriage returns and linefeed characters influence formatting, and acknowledge (ACK) and negative acknowledge (NAK) characters are used for communication.

There are various methods used for coding characters into binary. The most common methods are ASCII (American Standard Code for Information Interchange) and Unicode. In the past other coding systems such EBCDIC (Extended Binary Coded Decimal Interchange Code) were used. Note: EBCDIC is pronounced *eb-sih-dik*. Unicode is the current standard as it includes characters and symbols for all current languages. In addition Unicode includes all of the original ASCII character set. In the next section, we examine ASCII and Unicode in more detail.



GROUP TASK Discussion
Unicode is the standard method of representing characters on the Web. What if different coding systems were in common usage? What problems would this present to software developers? Discuss.

ASCII Code

ASCII is an acronym for American Standard Code for Information Interchange and is pronounced *ass-kee*. ASCII is a seven-bit code, which means there are 128 different characters in the standard ASCII character set.

Many extensions to the standard ASCII character set emerged over the years; until recently most of these were 8 bit codes. By extending the code from 7 to 8 bits, another 128 characters can be represented. Various different extended ASCII systems arose during the 1980s until the ISO Latin 1 system became the defacto standard. ISO Latin 1 is another 8-bit system that includes the standard ASCII character set. The International Standards Organization developed this standard for use on the World Wide Web. All these 8-bit systems use values above 127 to represent various foreign language, graphical and mathematical characters. For example, the character ½ in ISO Latin 1 uses code 189. The ISO Latin 1 codes together with ASCII are incorporated within the modern Unicode system. Therefore, code 189 in Unicode also represents the character ½. The standard ASCII codes shown in *Fig 10.10* are correct for both ISO Latin 1 and Unicode.

Standard ASCII has been developed to simplify the process of sorting. Notice that the code for 'A' is less than 'B' is less than 'C' and so on. Sorting alphabetically can be accomplished using the numerical ASCII codes. 'A' has an ASCII value of 65 or 1000001 in binary, 'a' has a code of 97 or 1100001 in binary. Each capital letter has an ASCII code 32 less than its lower case equivalent. This difference of 32 affects the value of the second highest order bit from a 0 to a 1.



Consider the following:

The word 'HELLO' coded in ASCII is 72, 69, 76, 76, 79. The 7 bit ASCII equivalent is 1001000, 1000101, 1001100, 1001100, 1001111.

The lower case word 'hello' can be similarly coded as 104, 101, 108, 108, 111. In binary, we get 1101000, 1100101, 1101100, 1101100, 1101111.

Notice that the last 5 bits of each character are the same in upper and lower case.



GROUP TASK Discussion
Explain how the above situation can assist when sorting and searching strings of characters.

Char	Dec	Char	Dec
NUL	0	@	64
SOH	1	A	65
STX	2	B	66
ETX	3	C	67
EOT	4	D	68
ENQ	5	E	69
ACK	6	F	70
BEL	7	G	71
BS	8	H	72
HT	9	I	73
LF	10	J	74
VT	11	K	75
FF	12	L	76
CR	13	M	77
SO	14	N	78
SI	15	O	79
DLE	16	P	80
DC1	17	Q	81
DC2	18	R	82
DC3	19	S	83
DC4	20	T	84
NAK	21	U	85
SYN	22	V	86
ETB	23	W	87
CAN	24	X	88
EM	25	Y	89
SUB	26	Z	90
ESC	27	[	91
FS	28	\	92
GS	29	]	93
RS	30	^	94
US	31	_	95
Space	32	`	96
!	33	a	97
"	34	b	98
#	35	c	99
$	36	d	100
%	37	e	101
&	38	f	102
'	39	g	103
(	40	h	104
)	41	i	105
*	42	j	106
+	43	k	107
,	44	l	108
-	45	m	109
.	46	n	110
/	47	o	111
0	48	p	112
1	49	q	113
2	50	r	114
3	51	s	115
4	52	t	116
5	53	u	117
6	54	v	118
7	55	w	119
8	56	x	120
9	57	y	121
:	58	z	122
;	59	{	123
<	60	|	124
=	61	}	125
>	62	~	126
?	63	DEL	127

Fig 10.10
The ASCII character set.

Unicode

Unicode specifies a unique number for every character in every language. Prior to the introduction and acceptance of Unicode in the 1990s there were many different coding systems in use for different languages. This meant that many codes referred to quite different characters and hence global communication of text in electronic form was difficult and often involved complex conversion. Currently Unicode is the global standard for coding characters. New characters and languages, including ancient scripts, are continually being added to the Unicode standard. The current standard including all coded characters is available to all at www.unicode.org. Once a code is assigned to a particular character then it will never change. This guarantee means any text coded using Unicode will be readable forever by any Unicode aware software.

To represent every current and future character in any language clearly requires a large and complex system. The Unicode system is based on a total of 17 code planes. Each code plane is 16-bits long; therefore each plane can include up to 2^{16}=65536 different characters. A total of around 1 million unique characters can be represented. The first plane, plane 0, is called the Basic Multilingual Plane or BMP. The 65536 code points on the BMP include all the characters for most modern languages in use today. The remaining 16 code planes are largely unused apart from some ancient languages and various specialist symbols.

Within the 16-bit Basic Multilingual Plane (BMP) are all the ASCII characters whose code points are identical to their corresponding ASCII codes. For example, an upper case "A" uses code 65 (or 41_{16}) in both ASCII and Unicode. Many of the earlier ISO Latin 1 codes are also identical. Furthermore many characters from other languages have been assigned codes so that conversion between older systems and Unicode is straightforward.



GROUP TASK Practical Activity
In Microsoft Word any character can be displayed by entering its Unicode hexadecimal code point value followed by Alt-x. Type your name using this "Alt-x" system of data entry.

The Unicode standard recommends that code points should be specified using "U+" followed by the hexadecimal value of the character, so capital "A" is specified as "U+0041". As the BMP is plane 0 then 16-bit values (4 hexadecimal digits) are assumed to be from the BMP. Characters from planes 1 to 16 are expressed using an extra one or two hexadecimal digits. For example "U+1F0A1" through to "U+1F0DF" are characters representing playing cards, i.e. through to . The leading 1 means these codes are found on plane 1.

There are three Unicode standards for representing text namely, UTF-8, UTF-16 and UTF-32, which unsurprisingly use 8-bits, 16-bits and 32-bits respectively to represent characters. UTF-8 is within the majority of web page files, UTF-16 is widely used within many desktop applications and UTF-32 is currently rarely used. At first glance these usage comments may appear strange as the range of Unicode characters is from 0_{16} up to $10FFFF_{16}$. The maximum value of $10FFFF_{16}$ includes 6 hexadecimal digits which correspond to 24-bits so surely 8-bits or 16-bits are insufficient? Clearly the encoding system cannot use just 8-bits or even 16-bits to represent all the possible Unicode code points. To overcome this problem both UTF-8 and UTF-16 use multiples of 8-bits (or 16-bits) to represent many of the less commonly used and numerically larger code points. This means the more common characters can be stored much more economically than is the case with UTF-32.



GROUP TASK Research Activity
Just because Unicode specifies a code point for a character does not mean that every font will include that character. Research and locate a font which includes the playing card characters U+1F0A1 through to U+1F0DF referred to above.



Consider UTF-8 encoding:

The Unicode code points range from 0_{16} up to $10FFFF_{16}$. UTF-8 encoding requires from one to four bytes to represent each possible Unicode code point. All the ASCII characters require a single byte and their codes are identical to the corresponding ASCII code which is a significant advantage over UTF-16.

Unicode range	Binary range of code point	Number of bits in code point	UTF-8 encoding
U+0000 to U+007F	0 to 1111111	1 to 7	0xxxxxxx
U+0080 to U+07FF	10000000 to 11111111111	8 to 11	110xxxxx 10xxxxxx
U+0800 to U+FFFF	100000000000 to 1111111111111111	12 to 16	1110xxxx 10xxxxxx 10xxxxxx
U+10000 to 10FFFF	10000000000000000 to 10000111111111111111	17 to 21	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

Fig 10.11
Unicode UTF-8 encoding

Notice that each of the multiple byte UTF-8 encodings begins with the same number of 1s as there are bytes in the encoding. This feature allows software to determine the byte length of each encoded character as it is being read. In addition all subsequent bytes in a multi-byte encoding commence with 10. This removes all ambiguity as all bytes beginning with a zero are single bytes (same as ASCII), all bytes commencing with 11 are the start of a multi-byte encoding and all bytes commencing with 10 are subsequent bytes within a multi-byte encoding.

- An upper case "A" is code point U+0041 (65 in decimal) which is 1000001 in binary and would be stored in a UTF-8 file as the single byte 01000001_2 which is simply 41_{16}.
- The Greek letter pi "π" is code point U+03C0 (960 in decimal) which is in the range U+0080 to U+07FF and is 1111000000 in binary. From *Fig 10.11* the required UTF-8 encoding is 110xxxxx 10xxxxxx which includes 11 x symbols, however there are 10 digits in 1111000000_2 so we need to add a leading 0 to make the required 11 x symbols. Therefore the character "π" in UTF-8 is represented by 11001111 10000000_2 which is $CF80_{16}$.



GROUP TASK Practical Activity
We noted earlier that U+1F0A1 through to U+1F0DF are characters representing playing cards. Determine the code point for the Ace of hearts and convert this value to its UTF-8 representation.



Consider UTF-16 encoding:

The Unicode code points range from 0_{16} up to $10FFFF_{16}$. UTF-16 encoding uses either one or two 16-bit sequences to represent each Unicode code point. In UTF-16 all the plane 0 (BMP) characters are simply represented identically to their code point. For instance U+0041 is represented by the 16-bits 00000000 01000001_2 which is 0041_{16}. Code points in planes 1 through to 17 require a pair of 16-bit codes or a total of 32-bits per character. These 16-bit pairs are known as surrogates. The first is called the lead (or high) surrogate and the second is called the trail (or low) surrogate.

Notice that for every ASCII character in UTF-16 the leading byte is a sequence of eight zeros. Clearly this is an inefficient use of storage, particularly for files which contain only ASCII text. However, for processing purposes this issue is minor as for virtually all current CPUs and RAM modules 16-bits is the smallest addressable unit. Even if UTF-8 encoding is used then ultimately 16-bits per character are utilised during processing.

Let us consider the system used to calculate UTF-16 surrogate values for code planes 1 to 17. These Unicode code points are in the range U+10000 to U+10FFFF, therefore we can immediately subtract 10000_{16} from each code point and save one hexadecimal digit (4-bits). We now have values in the range 0_{16} to $FFFFF_{16}$ which in binary requires 20-bits. If we split the 20-bits into two equal parts then each part will contain 10-bits in the range 0_{16} to $3FF_{16}$ as 1111111111_2 (ten binary ones) is equal to $3FF_{16}$. The high 10-bits are added to $D800_{16}$ to form the lead surrogate and the remaining low 10-bits are added to $DC00_{16}$ to form the trail surrogate. A few example calculations will help to explain the conversion process.

- Both upper case "A" (code point U+0041) and the Greek letter pi "π" (code point U+03C0) are in code plane 0 as they are less than U+10000. Hence in UTF-16 they are simply represented by their code point values 0041_{16} and $03C0_{16}$ respectively.
- The Queen of Spades character is code point U+1F0AD. As U+1F0AD is greater than U+10000 we need to calculate surrogates.
  - First we subtract 10000_{16} from $1F0AD_{16}$ which leaves $F0AD_{16}$.
  - We convert $F0AD_{16}$ into binary which gives us 1111000010101101_2.
  - We need this as a 20-bit number so we add leading zeros resulting in 00001111000010101101_2.
  - Split 00001111000010101101_2 into two to create the 10-bit numbers 0000111100_2 and 0010101101_2. Which in hexadecimal is $03C_{16}$ and $0AD_{16}$.
  - Add $03C_{16}$ to $D800_{16}$ to form the lead surrogate $D83C_{16}$.
  - Add $0AD_{16}$ to $DC00_{16}$ to form the trail surrogate $DCAD_{16}$.
  - Therefore U+1F0AD is represented in UTF-16 by $D83CDCAD_{16}$ which converts to the 32-bit binary number 1101100000111100 1101110010101101_2.



GROUP TASK Discussion
Code points U+D800 to U+DFFF in plane 0 are reserved and are never to be used for nay characters. Can you explain why? Discuss.



GROUP TASK Practical Activity
Convert Unicode code points U+03B1 and U+1F692 into UTF-16. What characters do these code points represent?

SET 10A

1. $(1 \times 2^5) + (1 \times 2^4) + (1 \times 2^1) + (1 \times 2^0)$ can be written in binary as:
   (A) 1111
   (B) 00011011
   (C) 110011
   (D) 00001111
2. $(1 \times 2^5) + (1 \times 2^4) + (1 \times 2^1) + (1 \times 2^0)$ has the decimal value:
   (A) 51
   (B) 48
   (C) 4
   (D) 25
3. When counting in binary the next number after 100111 is:
   (A) 101111
   (B) 100100
   (C) 101000
   (D) 110000
4. The Unicode code point U+1F0DE is represented in UTF-8 as which of the following binary values?
   (A) 11101111 10100001 10101111
   (B) 11111000011011110
   (C) 11110000 10011111 10000011 10011110
   (D) 000011111000011011110
5. The Unicode code point U+1F0DE is represented in UTF-16 as which of the following hexadecimal values?
   (A) D83CDCDE
   (B) 3C0DE
   (C) F0DE
   (E) D8F0DCDE
6. The decimal number 255 is equal to the hexadecimal number:
   (A) F0
   (B) 11111111
   (C) FF
   (D) FFF
7. The decimal number 83 is the equal to the binary number:
   (A) 1010011
   (B) 110011
   (C) 101011
   (D) 10100111
8. 110101110011 in binary is equal to the hexadecimal number:
   (A) D73
   (B) E73
   (C) 1373
   (D) 37D
9. "Frog" is represented in ASCII as:
   (A) 70, 82, 79, 71
   (B) 70, 114, 111, 103
   (C) 102, 114, 111, 103
   (D) 70, 114, 48, 71
10. The lower case word 'bird' is coded as 1100010, 1101001, 1110010, 1100100 in 7 bit binary ASCII. In upper case it would be coded as:
    (A) 1100010, 1101001, 1110010, 1100100
    (B) 11100010, 11101001, 1100010, 1110100
    (C) 0100010, 0101001, 0110010, 0100100
    (D) 1000010, 1001001, 1010010, 1000100
11. Binary numbers are used to represent data within computers. Wouldn't it be simpler if they used decimal numbers? Discuss reasons why computers use the binary number system rather than the decimal system.
12. Convert each of the following numbers into decimal:
    (a) 11101011_2 (b) $5D_{16}$ (c) 10111001_2 (d) &HAB (e) 1010101_2
13. Convert each of the following numbers into binary:
    (a) 11_{10} (b) 36_{16} (c) 96_{10} (d) 0x4B (e) 456_{10}
14. Convert each of the following numbers into hexadecimal:
    (a) 11101001_2 (b) 50_{10} (c) 816_{10} (d) 1000_{10} (e) 100111101_2
15. Imagine you have been commissioned to design a new coding system for the representation of characters on computers. Your new system uses 7 binary digits and must comply with the following requirements:
    - The digits 0 to 9 must be represented by their binary number equivalents. For example, the character 1 must be represented by 0000001.
    - Alphabetic characters must be in the order A, a, B, b, C, c… Z, z.
    - There must be a link between each character and their QWERTY keyboard shift equivalent. For example ":" is produced on a standard QWERTY keyboard by holding down shift and pressing ";", these characters must be linked somehow in your coding system.
    - Control characters must have larger values than any of the printable characters.

    Create a coding system that complies with all the above requirements.

INTEGER REPRESENTATION

Integers are whole numbers, both positive and negative. For example, 456, 12, 0, -1 and -3965 are all integers, 3¼, π, -2.995, 0.888… and √2 are not integers, rather they are real numbers. Integers form a subset of the real numbers. Zero is a special integer; it is neither positive nor negative.

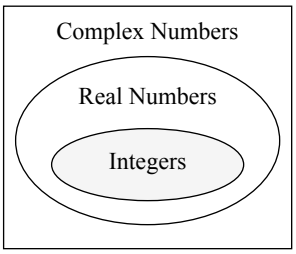


Fig 10.12
Integers are a subset of the real numbers, which in turn are a subset of the complex numbers.

We have already seen how positive integers and zero can be represented using binary on computers. How do we represent negative integers using only zeros and ones? We can't use a '-' sign directly as computers can only work with zeros and ones. A number of techniques have been developed. The aim of each technique is to provide a system that allows accurate and efficient calculations to take place.

In this section, we examine two integer representation techniques:

- Sign and modulus.
- Two's complement (and one's complement).

Sign and modulus

The sign and modulus method is perhaps the most obvious solution. We just add an extra bit to the front of each number to represent the sign of the number. A zero is used for positive numbers and a one for negative numbers. The modulus is the magnitude of the number or the absolute value of the number. Mathematically the absolute value of a number is the distance from the number to zero.

If we are using 7 bits for the modulus then the number 12 would be represented as 00001100 and -12 as 10001100. Using this system, zero can be represented in two ways; either 10000000 or 00000000.



Consider the following:

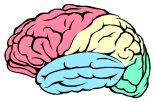
Using the sign and modulus method, the sign of the number is not an integral part of the number. This means that calculations must take account of the sign bit separately to the actual calculation. This is essentially what we do when adding or subtracting negative numbers by hand. For example, when adding say 12 and -7 we know that adding a negative is the same as subtracting, so we perform the calculation 12 – 7 and get the result 5. Computers find subtraction unnecessary and hence the sign and modulus method is not generally used on modern computers.



GROUP TASK Discussion
The sign and modulus technique is difficult to implement on a computer because computers are not real smart. They just like to add things, mind you they can add at lighting speed, subtraction they find difficult and unnecessary. Can you think of any integer subtractions that cannot be rewritten as additions? Is subtraction really necessary? Discuss.

Two's complement (and one's complement)

Working out a method of representing integers where the sign of the number is included as an integral part of the representation is solved using complements. Let us discuss the thinking behind this method using the decimal system before we implement it in binary.



Consider the following:

In primary school, you most likely used a number line, in Year 7 this number line was extended to include negative numbers. Mathematically the number line extends infinitely in both positive and negative directions. For a moment, imagine we dispense with the negative or left hand side of the number line and that our positive numbers are finite, say with a maximum value of 1000. Imagine we join the ends of this finite number line to form a circle (see *Fig 10.13*).

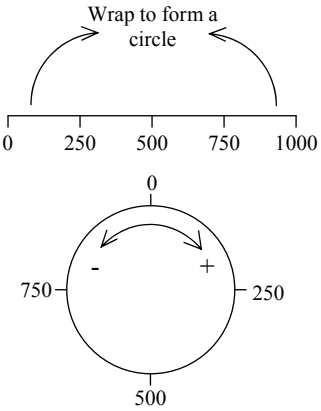


Fig 10.13
Wrapping a finite positive number line to form a circle.

Now if we move in a positive or clockwise direction our integers increase in value. If we move in a negative or anti-clockwise direction our integers decrease in value, with one important exception, zero. Moving anti-clockwise from zero we would like to decrease into the negative numbers, -1, -2, -3 and so on, instead we reach 999, 998, 997… These numbers are complements of their positive equivalents, in our example the ten's complement of 1 is 999, of 2 is 998, of 3 is 997 and so on. Each of these pairs of numbers is the same distance from zero on our circular number line.

We cannot have 999 representing both -1 and 999 so we need to develop some rules. Let us make sure there are an equal number of positive and negative integers. Positive's run from 0 to 499 and negatives from 999 down to 500.

Now let us test our results using normal addition. For example -3 + -5, using our ten's complement system this becomes 997 + 995 giving us a result of 1992. If we discard the leading 1, we get the correct answer 992, which using our system is equivalent to -8. Let us try -6 + 19, this becomes 994 + 19 resulting in 1013. Discarding the leading 1 gives us the correct result 13.



GROUP TASK Activity
Perform the following additions using our number system developed above: 18 + -25, -100 + 109, -5 + 5 and -15 + -39. Make sure you understand this system thoroughly before moving onto the next section.



GROUP TASK Discussion
Remember the aim was to develop a system where no sign is required for negative numbers. How has this been achieved? Does the above number system work with subtraction, multiplication and division?

We now examine this system as it is applied to binary numbers. Generally, integers are stored within computers using 16, 32 or 64 bits. For our discussion, we will use 8 bits to help simplify the mathematics. With 8 bits, we can represent numbers from 00000000 up to 11111111. If we form a circular number line using these numbers, we see that 11111111 should be used to represent -1. *Fig 10.14* shows the significant part of this circular number line on either side of zero.

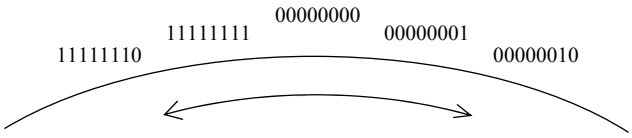


Fig 10.14
Section of an 8 bit circular number line.

Let us examine pairs of positive and negative numbers using this system. *Fig 10.15* shows the first sixteen integers along with their two's complement. For example, the two's complement of 00000001 is 11111111.

Positive	
Decimal	Binary
1	00000001
2	00000010
3	00000011
4	00000100
5	00000101
6	00000110
7	00000111
8	00001000
9	00001001
10	00001010
11	00001011
12	00001100
13	00001101
14	00001110
15	00001111
16	00010000

Negative	
Decimal	Binary
-1	11111111
-2	11111110
-3	11111101
-4	11111100
-5	11111011
-6	11111010
-7	11111001
-8	11111000
-9	11110111
-10	11110110
-11	11110101
-12	11110100
-13	11110011
-14	11110010
-15	11110001
-16	11110000

Fig 10.15
Two's complements for the first sixteen integers.

A number of patterns emerge. Firstly, each positive number commences with a 0 and each negative with a 1. In fact, it turns out that this is always the case regardless of the number of bits used.

Secondly, exchanging 1s for 0s and 0s for 1s almost results in the correct two's complement, in fact the result is out by 1. For example, 7 is 00000111, swapping 1s and 0s gives 11111000, which has a value of -8. If we add 1 to this result, we get the correct answer 11111001.

This process of swapping 1s and 0s results in the one's complement of the number. Why is it called the one's complement? Because the swapping of 1s and 0s can be achieved by subtracting the number from a string of 1s. For example, 11111111 – 00110111 results in 11001000, hence the name one's complement. It is possible to perform arithmetic using ones complement numbers, however the two's complement system proves to operate more efficiently.

Further analysis of our two's complement table in *Fig 10.15* reveals another simpler method of calculating two's complements. Begin reading the original binary number from the right hand side moving one digit at a time to the left. As soon as a one is encountered commence swapping 1s and 0s, just don't change the first one. For example, if we wish to find the two's complement of 12, which is 00001100 in binary. Scanning from the right hand side, we encounter a 1 in the third position from the right. Leave this 1 and then swap all the 1s and 0s to its left; the result is 11110100, which is correct!



GROUP TASK Activity
Extend the table in *Fig 10.15* to include all possible 8-bit integers that can be represented. What is the range of integers represented using 8 bits?



GROUP TASK Activity
Write down ten 8-bit two's complement numbers, 5 positive and 5 negative. Convert each number to its two's complement. Confirm that the system works to convert negatives to positives as well as positives to negatives.

We mentioned above that all positive two's complement numbers commence with a zero and all negatives with a one. This information is particularly useful if we wish to convert a two's complement number into decimal.

Using our 8-bit two's complement system -12 is represented as 11110100. Each binary digit has a place value, commencing from the right hand side with 2^0=1, 2^1=2, 2^2=4, 2^3=8, 2^4=16, 2^5=32, 2^6=64 and the last or left hand bit has a place value of -2^7= -128. The bit on the left hand side is often called the 'highest order bit' or the 'most significant bit'. We can write 11110100 as -128 + 64 + 32 + 16 + 0 + 4 + 0 + 0 which equals -12. If we were performing many of these conversions, it is convenient to draw up a table such as the one in *Fig 10.16*. Add up all the positive values first and, if necessary, subtract 128 at the end.

-128	64	32	16	8	4	2	1	Decimal
1	1	1	1	0	1	0	0	-12
0	1	0	1	1	0	1	0	90
1	0	0	0	0	1	0	1	-123
0	0	0	0	1	1	0	1	13

64	64			
32	16			8
16	8	4		4
4	2	1		1
116	90	5		13
- 128		- 128		
- 12		- 123		

Working

Fig 10.16
Table for converting two's complement to decimal.



GROUP TASK Activity
Draw up a table, like the one in *Fig 10.16*. Randomly fill in the table with ones and zeros. Convert each number to its decimal equivalent.



GROUP TASK Discussion
Most computers use 16, 32 and/or 64-bit two's complement to represent integers. What is the range of integers that can be represented using each of these three representations?

REPRESENTATION OF FRACTIONS AND REAL NUMBERS

Computers cannot represent all the possible real numbers precisely. We have the same problem when writing decimals; we cannot precisely write the fraction 1⅓ as a decimal. However, we can write it as accurately as we wish. For example, 1·33333 is a better approximation than 1·3. What we require is a system that will provide us with a satisfactory level of accuracy for the majority of applications. Note that our two's complement system is able to represent integers precisely, albeit within a limited range.

There are two methods used for representing fractions; fixed-point and floating-point. Both these methods are similar to the method we use to write decimal fractions. For example, 475·382 means 4 hundreds, 7 tens, 5 units, 3 tenths, 8 hundredths and 2 thousandths. In expanded form, we write 475·382 as $(4 \times 10^2) + (7 \times 10^1) + (5 \times 10^0) + (3 \times 10^{-1}) + (8 \times 10^{-2}) + (2 \times 10^{-3})$. We can do the same with binary fractions. For example, 101·1101 means 1 four, 0 twos, 1 unit, 1 half, 1 quarter, 0 eighths and 1 sixteenth or in expanded form as $(1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) + (1 \times 2^{-1}) + (1 \times 2^{-2}) + (0 \times 2^{-3}) + (1 \times 2^{-4})$. This number in decimal is $4 + 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{16} = 5\frac{13}{16} = 5 \cdot 8125$.

Let us now examine the difference between fixed-point and floating-point representations.

Fixed-point

Fixed-point refers to the position of the decimal point. When using fixed-point numbers the decimal point remains in the same position for all numbers. For example, a fixed-point system using 16 bits may have 8 bits for the whole number part and 8 bits for the fractional part. The problem with such a system is that very large numbers and very small numbers cannot be represented. The smallest positive number that can be represented using two's complement and 16 bits is $0.00000001_2 = 0.00390625_{10}$ and the largest is $01111111.11111111_2 = 127.99609375_{10}$.

Fixed-point numbers are not included as standard data types in most high-level languages; rather, they are created by programmers using one of the integer data types. Fixed-point representations are generally faster and more accurate than floating-point, unfortunately for most applications their range is insufficient so floating-point numbers are used. Many specialised applications where the range of numbers is small yet accuracy is important will used fixed-point representations. The fixed-point method is essentially the same as the two's complement system with the decimal point shifted to an appropriate position for the particular application.

Floating-point

Floating-point numbers can be thought of as generalised fixed-point numbers. The position of the decimal point is encoded within the format. In this way, the decimal point can be placed in a different position within the number. Essentially, the number is stored using scientific notation. For example, in decimal 2345.678 can be written as 2.345678×10^3, which is often written as +2.345678E+3. There are three essential components to each number; the mantissa which in this case is 2.345678, the exponent which is 3 and the sign of the number, in this case +. The value of the exponent determines the position of the decimal point. The floating-point system allows us to represent a much larger range of numbers using less storage space. For example, 0.000000000000000000000052 can be written as +5.2E-23.

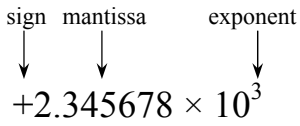


Fig 10.17
Three components necessary for specifying a floating-point number.

Working with floating-point numbers, compared to integers, is a complicated and processor intensive process. Today's microprocessors contain a special maths coprocessor called a Floating-point Unit or FPU. The FPU is dedicated to the processing of floating-point numbers.

The Institute of Electrical and Electronics Engineers (IEEE) has produced a standard for representing binary numbers on computers using the floating-point system. IEEE Standard 754 specifies two formats; single precision, using 32 bits, and double precision, using 64 bits. All the major hardware and software developers have accepted these standards.



GROUP TASK Discussion
Fixed-point calculations are supposedly less processor intensive that floating-point calculations. Why do you think this is the case? Use decimal examples to explain your answer.

IEEE 754 single precision floating-point standard.

This standard requires 32 bits, the first bit is the sign bit, the next eight bits are used to store the exponent and the final twenty-three bits are the fractional part. *Fig 10.18* shows the binary number -1.11001×2^{1101} represented using this standard. Let us examine each component in turn.

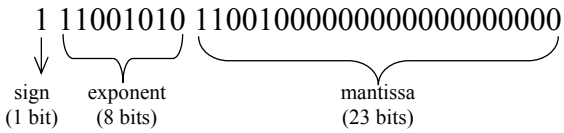


Fig 10.18
The IEEE single precision floating-point standard uses 32 bits to represent each number.

The sign bit is simply set to 0 if the number is positive and 1 if the number is negative. As -1.11001×2^{1101} is a negative number, the sign bit is set to 1.

The actual value of the exponent can be either negative or positive. This presents a problem, how do we represent the sign of the exponent most efficiently? We could use one bit of the 8 bit exponent to represent its sign, we could use the two's complement system or is there a better way? The standard specifies that the exponent is represented as the actual value plus 127. Because we have 8 bits for the exponent, we can represent integer values from 0 to 255. Subtracting 127 gives us a more useful actual range from -127 to 128. In our -1.11001×2^{1101} example, the exponent is 1101_2 which in decimal has a value of 13. If we add 127 to 13 we get 140, 140 in binary is 11001010_2.

The mantissa is essentially the same as the original number with one modification. Because we are working in binary, the first digit, to the left of the decimal point, will always be a one. The IEEE standard specifies that a leading one is always assumed. In our -1.11001×2^{1101} example, the unsigned mantissa is 1.11001; we ignore the leading 1 so our mantissa becomes 11001.

Let us consider the range of numbers that can be represented. Initially, we would assume the largest number to be represented by a zero followed by 31 ones and the smallest negative using 32 ones. The IEEE 754 standard includes a number of special cases, which give special meanings to numbers with exponents containing eight ones. As a consequence, the largest exponent is 11111110. Let us calculate the decimal equivalent of the range:

- Largest = 0 11111110 11111111111111111111111 (IEEE 754 Single)
- Smallest = 1 11111110 11111111111111111111111 (IEEE 754 Single)
- The exponent is made up of 7 ones and a zero, 11111110 is 254 in decimal. Now 254 – 127 is 127, the decimal exponent.
- The mantissa is a series of 23 ones, 11111111111111111111111. We need to add the leading one, giving the binary fraction 1.11111111111111111111111. This value is very close to the decimal number two.
- The largest number in decimal is therefore 2×2^{127} which is equivalent to 2^{128}. Using a calculator the largest number becomes approximately 3.4×10^{38}, therefore the range of numbers that can be represented is approximately -3.4×10^{38} to 3.4×10^{38}.

There are a number of mathematical operations that cannot be computed, for example division by zero and finding the square root of negative numbers. We also require a method for representing infinity and negative infinity. As a result of not including the leading one in the mantissa, we also require a special case for zero. These special cases have been included in the IEEE 754 standard. *Fig 10.19* shows the representations for each of these cases.

Infinity	0 11111111 00000000000000000000000
Negative Infinity	1 11111111 00000000000000000000000
Zero	0 00000000 00000000000000000000000
Negative Zero	1 00000000 00000000000000000000000
Not a Number	x 11111111 xxxxxxxxxxxxxxxxxxxxxxx

Fig 10.19
Special cases in the IEEE 754 floating-point standard.

The not a number case is used for meaningless operations such as division by zero and square roots of negative numbers. In these cases, an error code is often coded into the mantissa part of the number. This allows feedback to be returned to programmers on the nature of the meaningless operation.

IEEE 754 double precision floating-point standard.

This standard uses 64 bits to represent each number. A sign bit, 11 bits for the exponent and 52 bits for the mantissa.

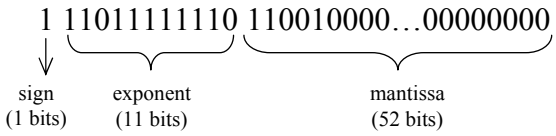


Fig 10.20
The IEEE double precision floating-point standard uses 64 bits to represent each number.

The extra exponent bits extend the range of double precision numbers to approximately -10^{308} to 10^{308} base 10. Also, to enable the representation of negative exponents, we represent the exponent as the actual value plus 1023. The extra mantissa bits increase the accuracy of computations. Of course, these benefits require extra processing power. The remainder of the IEEE 754 double precision standard is the same as the single precision standard.



GROUP TASK Activity
Write down the smallest positive number using the IEEE 754 single precision standard. Convert this number to a decimal. Perform the same process using the smallest positive double precision number. Compare these two results.



GROUP TASK Activity
Write down five single precision floating-point numbers. Convert each number to its decimal equivalent.



GROUP TASK Discussion
1023 is added to the actual value of the exponent when using the IEEE 754 double precision format. Explain why this is the case?

SET 10B

1. Which of the following is NOT an integer?
   (A) 55034
   (B) -38933
   (C) -18.5
   (D) 0

2. Using 8 bit sign and modulus representation the integer value -1 is represented as:
   (A) 11111111
   (B) 10000001
   (C) -00000001
   (D) 00000001

3. The 8 bit two's complement of 55 is:
   (A) 00110111
   (B) 00111000
   (C) 11001001
   (D) 11001000

4. The main difference between fixed-point representations and floating-point representations of real numbers is that:
   (A) with fixed-point the decimal point can change position whereas with floating-point there is no decimal point.
   (B) fixed-point uses two's complement for the mantissa whereas floating-point does not.
   (C) floating-point allows the decimal point to move to any position, fixed-point does not.
   (D) fixed-point numbers can only represent positive numbers whereas floating-point numbers can represent both positive and negative numbers.

5. Using 16 bit two's complement the range of integers that can be represented is:
   (A) -128 to 127
   (B) -32768 to 32767
   (C) -65536 to 65535
   (D) 0 to 65535

6. 10101010 in 8 bit two's complement is equivalent to the decimal value:
   (A) -86
   (B) 86
   (C) 170
   (D) 42

7. The one's complement of 10101110 is:
   (A) 10101110
   (B) 10101111
   (C) 01010001
   (D) 01010010

8. In the number -6.78×10^5:
   (A) - is the sign, 6.78 is the exponent and 5 is the mantissa.
   (B) - is the sign, 6.78 is the mantissa and 5 is the exponent.
   (C) - is the exponent, 6.78 is the mantissa and 5 is the base.
   (D) 10 is the exponent, 6.78 is the exponent and 5 is the mantissa.

9. The IEEE single precision floating-point standard uses:
   (A) 1 bit for the sign, 8 bits for the exponent and 23 bits for the mantissa.
   (B) 1 bit for the sign, 11 bits for the exponent and 52 bits for the mantissa.
   (C) 1 bit for the sign, 4 bits for the exponent and 11 bits for the mantissa.
   (D) 1 bit for the sign and 16 bits for the modulus

10. Real numbers cannot always be represented precisely using computers. This is because:
    (A) computers are finite machines and some real numbers go on forever.
    (B) many real numbers cannot be represented exactly in binary.
    (C) computers represent real numbers using a system that is a compromise between accuracy and processing/storage ability.
    (D) All of the above.

11. Integers are generally represented within computers precisely, where as real numbers are not. Explain why this is the case.

12. The two's complement system is commonly used to represent integers within computers where as the sign modulus system is rarely used. Explain why this is the case?

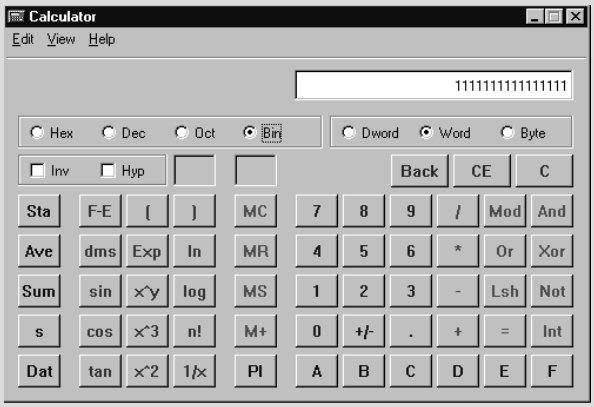
13. Convert each of the following integers into their 16 bit two's complement equivalent:

    (a) 0 (b) 1 (c) -1 (d) 345 (e) -580

14. Convert each of the following decimal numbers into their binary IEEE 754 single precision floating-point equivalent:

    (a) 0.00015 (b) 4800000 (c) -1 (d) -0.0005 (e) -3.2×10^{18}

15. The screen shot at right is taken from the calculator application included with versions of the Microsoft Windows operating system. The binary Word number system on this calculator, uses 16-bit two's complement. However there are problems with this implementation of the twos complement system when converting to other number systems.



    The number on the screen is a series of sixteen ones. If we click on the 'Dec' option button, the number is converted to 65535_{10}. If we then add 1 to the number, we get 65536 displayed, converting this back to binary by clicking the 'Bin' option results in 0 being displayed. Also, if we are in binary and perform the subtraction 0 – 1 then sixteen ones are displayed as the answer, converting this to decimal again displays 65535.

    How can you explain the above results? What checks could have been included to correct these anomalies?

BINARY ARITHMETIC

The four operations of addition, subtraction, multiplication and division can be accomplished in binary using similar techniques as those used with decimals. Because we have only two digits in binary, 1 and 0, these operations become simpler, we need only know number facts involving 0 and 1. Unfortunately, performing operations on large numbers becomes somewhat tedious, as binary numbers tend to be much longer than their decimal equivalents.

In this section, we will examine the four arithmetic operations using two's complement integers. Similar processes are involved when using fixed and floating-point real numbers, however multiple computations are required. For example, consider multiplying two decimal numbers written in scientific notation; the mantissas are multiplied, the exponents are added, then the result is normalised so the mantissa is between 0 and 10. Multiplication of floating-point numbers involves similar processes.

Let us examine each of the four operations in turn:

Binary addition

When we first learnt addition in primary school, we were taught all our basic number facts with numbers up to 10. For example, 5 + 3 = 8, 6 + 9 = 15, 0 + 5 = 5, etc…; 100 different number facts in all. In binary, we need only learn a total of five number facts. In fact, the only binary addition number facts required are 0 + 0 = 0, 0 + 1 = 1, 1 + 0 = 1, 1 + 1 = 10 and 1 + 1 + 1 = 11.

+	0	1
0	0	1
1	1	10

1 + 1 + 1 = 11

Fig 10.21
Binary addition number facts.

The method of performing addition in decimal can also be used for binary addition. Each column is added commencing with the right hand side. If the result is 10 or 11 then a 1 is carried to the next column.



Consider the following:

Using 8 bit twos complement perform the following additions:

1. 01010111 + 00011100
2. 01011101 + 01110100
3. 11000000 + 01110000

1. Let us examine the working for this question in detail.

01010111	+ 01010111	+ 01010111	+ 01010111	+ 01010111	+ 01010111	+
00011100	00011100	00011100	00011100	00011100	00011100	
1	11	011	0011	10011	01110011	
1+0=1	1+0=1	1+1=10 write 0 carry 1	1+0+1=10	1+1+1=11	1+0+0=1, 1+0=1, 0+0=0	

The method, or algorithm used to perform binary addition is the same as that used for decimal addition. Remember that 10_2 is the same as the decimal value 2 and 11_2 is the same as the decimal value 3. The result 01110011 is positive, as the leading bit is a zero.

2. Applying the same technique used above we get the result 11010001 (see *Fig 10.22*). This result is a negative number in 8 bit two's complement, as the leading bit is a one. How can this be when the numbers in the question are both positive? Let us examine the question in decimal. $01011101_2 = 93_{10}$ and $01110100_2 = 116_{10}$, in decimal 93 + 116 = 209. 209 is outside the range of values available in 8 bit two's complement, so the result has wrapped around into the negatives. (8 bit two's complement has a range from –128 to 127). If this situation occurs in source code, an overflow run time error would result. Programmers need to be careful that the data type chosen is able to contain the results of any calculations within their source code.

```
  1 1 1 1 1
  01011101  +
  01110100
 ----------
  11010001
 ----------
```

Fig 10.22
An overflow error occurs if the result is outside the allowable range.

3. This question results in a carry in the final stage of the addition (see *Fig 10.23*). As we have only 8 bits to work with, we ignore this last carry. Ignoring the final carry, in this case, effectively causes the result to wrap around past zero. As the result is still within the range of our two's complement system no overflow error results and the correct answer is calculated. In decimal, this question is the equivalent of -64 + 112 = 48.

```
  1 1
  11000000  +
  01110000
 ----------
  00110000
 ----------
```

Fig 10.23
Ignore any final carry when using the two's complement system.

Let us summarise what we have learnt about two's complement addition from Questions 2 and 3:

- Calculations that move through zero will not result in errors, as the answer will still be within the allowable range. For example, -45 + 50 and 12 + -18.
- Calculations that result in values outside the range will result in overflow errors. For example, -90 + -45 and 105 + 35.
- Carries occurring in the final stage of an addition can always be ignored.

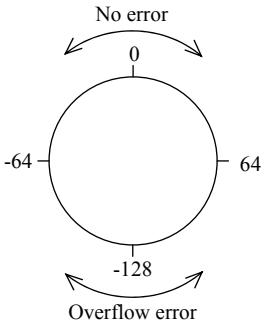


Fig 10.24
Two's complement addition special cases.



GROUP TASK Activity

Perform each of the following binary additions:

1.	01011111 +	2.	01110101 +	3.	11010000 +
	00001100		00110110		01100011

Assuming these numbers are expressed using 8 bit two's complement, will any errors occur? Perform each addition in decimal to confirm your results.



GROUP TASK Investigation

Examine the binary addition function available on your pocket calculator. Determine whether two's complement is used. How many bits are used? How are overflow errors reported?

Binary subtraction

Although binary subtraction can be performed in a similar way as we perform decimal subtraction, computers do not use it, rather we add the negative. For example 5 – 3 is the equivalent of 5 + ¯3. This process can be used to convert all subtractions to additions. When working with two's complement binary numbers we convert the number being subtracted to its two's complement and then add.



Consider the following:

Using 8 bit twos complement perform the following subtractions:

1. 01010111 - 00011100
2. 00011001 - 11111011

1. Let us examine the working for this question in detail. Firstly, we need to find the two's complement of the number being subtracted. In this case the number, 00011100, is positive as the leading bit is a zero, so we expect the two's complement to be negative. We can find the two's complement in two ways; find the one's complement and add one or scan from right to left and swap 1's and 0s after the first 1 is encountered.

   For this question, we will use the second method. Scanning 00011100 from the right hand side, we encounter a 1 in the third position, leave this 1 as is and change all the other digits. Our result is 11100100, which is negative as expected.

   Performing our addition, 01010111 + 11100100, gives us the result 00111011. In decimal, we have completed the subtraction 87 – 28 = 87 + ¯28 = 59.

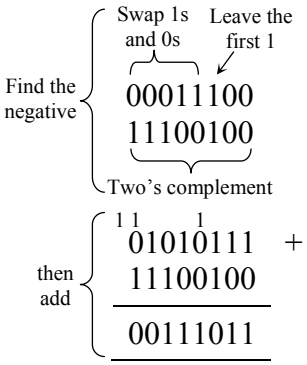


Fig 10.25
Subtraction is performed by adding the negative.

2. In this question, the number being subtracted is negative as the leading bit is a 1. We need to convert this number, 11111011, to its two's complement, which will be positive. Let us find the one's complement and add 1. The one's complement is found by swapping 1s and 0s. The one's complement of 11111011 is therefore 00000100. Adding 1 to this result gives us 00000101, which is positive as expected.

   Adding the two numbers 00011001 and 00000101 yields the result 00011110. In decimal we have performed the subtraction 25 - ¯5 = 25 + 5 = 30.



GROUP TASK Activity
Perform each of the following 8 bit two's complement binary subtractions:

1. 10010111 - 00001101
2. 01110001 - 00110110
3. 01010011 - 11110011



GROUP TASK Discussion
Microprocessors are not hardwired to perform subtraction. Why do you think this is the case?

Binary multiplication

The method used to multiply in binary is the same as that used for decimal. Fortunately, in binary we only have the two digits 0 and 1. Multiplying by 0 or 1 is certainly an easier proposition than multiplying by 0, 1, 2, 3, 4, 5, 6, 7, 8 or 9.

Let us consider a simple example: 1001 × 101, which in decimal is 9 × 5. The working for this multiplication is shown in *Fig 10.26*. Notice that each row in our working either contains all zeros or it contains the first number in the multiplication shifted zero or more places to the left. Binary multiplication is often called 'Shift and add'. In essence, a microprocessor need not know how to multiply if it is able to shift bits to the left and it can add.

```
   1001×
    101
  ------
   1001
  00000
 100100
 ------
 101101
 ------
```

Fig 10.26
Binary multiplication can be performed like decimal multiplication.

We examined a number of 8086 assembler instructions in chapter 5. 8086 assembler includes a SAL and a SAR statement, meaning 'Shift Arithmetic Left' and 'Shift Arithmetic Right', used in conjunction with the addition instructions ADD (Addition) and ADC (Addition with Carry) binary multiplication is accomplished. All microprocessors have the equivalent of these instructions hardwired as part of their micro-code instruction set.

When performing multiplication by hand we perform all our sub-multiplications first and then add them all up at the end. Computers perform a shift (the equivalent of a sub-multiplication) and add the result to a running total. They then perform another shift and add the result to the total, this process continues until the multiplication is complete. *Fig 10.27* shows an algorithm that could be used to complete this process. Binary multiplication when performed on computers is often called "shift and add".



Consider the following:

Let us perform a desk check of the algorithm in *Fig 10.27* to illustrate the processes performed by the CPU when multiplying binary numbers. Let us assume we are working with an 8 bit processor.

Using 1101 and 1001 as the inputs:

Product	Num1	Num2
00000000	00001101	00001001
00001001	00000110	00010010
	00000011	00100100
00101101	00000001	01001000
01110101	00000000	10010000

This is the equivalent of the decimal multiplication 13 × 9 = 117. In reality, the CPU multiplying process requires various other processes apart from those described.

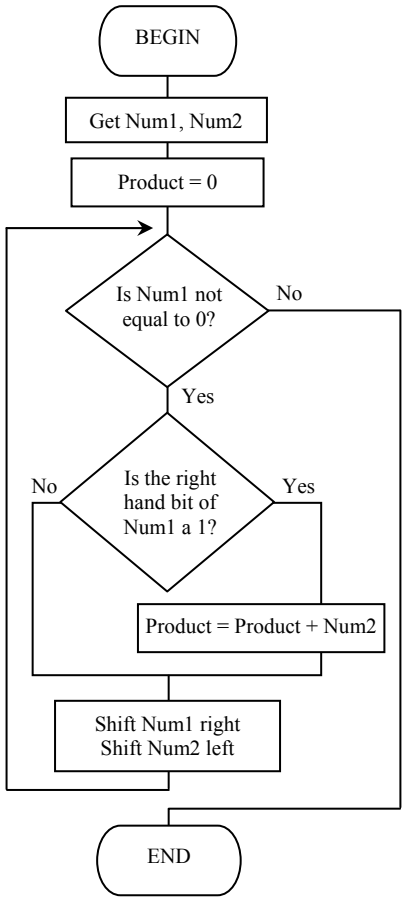


Fig 10.27
Algorithm describing binary multiplication using shifts and adds.



GROUP TASK Activity
Perform each of the following binary multiplications using the same method used for decimal multiplication:

1. 1010111 × 1101
2. 1110001 × 1110
3. 1010011 × 110011



GROUP TASK Activity
Using 16 bit two's complement multiply -20 and 27. Convert these numbers to binary and then use the algorithm in *Fig 10.27* to perform this operation. Confirm that your binary result is equivalent to -540.

Binary division

Division in binary can be performed using the same techniques used with decimals. Let us examine one common technique with a decimal example. We will then emulate this technique in binary.

Let us consider 6925 ÷ 7. The working for this division is shown in *Fig 10.28*. The result is 989 with a remainder of 2, for our purposes, we are interested in integer division so we will ignore the remainder.

```
   989
7 )6925
  63
  ---
   62
   56
   --
    65
    63
    --
     2
```

Fig 10.28 Decimal division example.

Firstly, we ask ourselves how many times 7 goes into 6, it doesn't, we could write a 0 up the top. We then ask does 7 go into 69, yes, 9 times, so we write a nine up the top. In fact, what we are doing is calculating the number of times 7 goes into 6900. The answer is 900, we write a 9 in the hundreds column. We then perform the subtraction 69 – 63, this equals 6. We then bring down the 2 from above and ask how many times 7 goes into 62, which is really how many times does 7 go into 620. The answer is 80 times so we write an 8 in the tens column. The process continues in this way until all digits have been brought down from above.

Now let us consider this technique with a binary example, 111010 ÷ 110. Does 110 go into 1, no, does 110 go into 11, no. We now ask does 110 go into 111, yes it goes in once, in fact it will either not go in or it will go in once. Because we are working in binary this is always the case. Subtracting 110 from 111 leaves us with 1. Actually, we are performing the binary subtraction 111000 – 110000, which is performed by computers by adding the negative. We bring down the next digit, a zero, and so on until we have a number greater than 110, namely 1010. Subtracting 110 leaves a remainder of 100. In decimal, we have performed the division 58 ÷ 6 = 9 remainder 4.

```
    1001
110)111010
   110
   ----
    1010
     110
     ---
     100
```

Fig 10.29 Binary division example.



GROUP TASK Activity
Perform each of the following binary divisions using the method described above:

1. 1010101 ÷ 1101
2. 1110101 ÷ 1010
3. 1010011 ÷ 101

Let us now consider the processes used by computers to perform division. Division is the most complex of the four basic arithmetic operations and there are many complex ways of performing binary division.

One common, yet complex, method involves the use of partial fractions. Using this method the division is viewed as a fraction. For example, 58 ÷ 6 is equivalent to 58/6. This fraction is broken down into a series of partial fractions with denominators that are all powers of two e.g. 58/6 ≈ 58/8 + 58/32 + 58/128 + 58/512 + 58/2048 + 58/8192. Evaluating each of these partial fractions in binary is straightforward. Dividing a binary number by 2 is accomplished by shifting 1 position to the right. Dividing by 4, shift 2 positions, dividing by 8, shift 3 positions, and so on. Adding up each of these partial results yields the answer to the original division. Calculating the partial fractions is a mathematical process that is beyond the scope of this course. Nevertheless, it is important to realise that this method exists and is often used.

Another common method of performing binary division on computers emulates the techniques used on the previous page to perform division by hand. *Fig 10.30* describes an algorithm that could be used to implement this technique. This technique is often called 'shift and subtract'. Initially the divisor is shifted to the left of the available bits. We then progressively shift the divisor to the right. After each shift, we compare its value with the remainder. If the remainder is larger, we subtract and record a one in the quotient. Eventually, the divisor will hold a value smaller than the original Num2 input, signifying the process is finished.

```
BEGIN BinaryDivision
    Get Num1,Num2
    Divisor = Num2
    WHILE Left hand bit of divisor is 0
        Shift Divisor left
    ENDWHILE
    Remainder = Num1
    Quotient = 0
    WHILE Divisor ≥ Num2
        Shift Quotient left
        IF Remainder ≥ Divisor THEN
            Remainder = Remainder – Divisor
            Quotient = Quotient + 1
        ENDIF
        Shift Divisor right
    ENDWHILE
END BinaryDivision
```

Fig 10.30
Algorithm for binary division.

Fig 10.31 shows a desk check of this algorithm using 111010 and 110 as the initial inputs. In other words, we are performing the decimal division 58 ÷ 6 in binary.

Initially Num1 is set to 00111010 and Num2 to 00000110 if we use an 8 bit system.

Divisor	Remainder	Quotient
00000110		
00001100		
00011000		
00110000		
01100000		
11000000	00111010	00000000
01100000		00000000
00110000		00000000
00011000	00001010	00000001
00001100		00000010
00000110		00000100
00000011	00000100	00001001

Fig 10.31
Desk check of the 'shift and subtract' binary division algorithm.



GROUP TASK
Activity

Perform a desk check of the 'shift and subtract' algorithm shown in *Fig 10.30* using 111100 and 101 as the inputs. This is the equivalent of 60 ÷ 5 in decimal.



GROUP TASK
Investigation

Dividing by powers of 2 is simple in binary. Do you agree with this statement? Explain using a number of examples.



HSC style questions:

Question 1.

A programming language defines the Byte data type as follows:

> **Byte Data Type**
> Unsigned 8-bit (1-byte) integer ranging in value from 0 through 255

The language also includes a Hex() function defined as follows:

> **Hex Function**
> Returns a string representing the hexadecimal value of a number

The following code in this language is executed:

```
10  Define A, B, C as Byte data type
20  A = 180
30  B = 9
40  C = A / B
50  Println Hex(A) + " divided by " + Hex(B) + " equals " + Hex(C)
```

(a) Line 10, 20 and 30 have executed. Determine the binary numbers held in memory representing the contents of variable A and variable B. Show all working.

(b) Manually perform the binary division the CPU would carry out to execute line 40. Show all working.

(c) Determine the output printed when line 50 is executed. Show all working.

(d) A new version of the same programming language redefines the Byte data type as follows:

> **Byte Data Type**
> 8-bit (1-byte) signed two's complement integer

Discuss possible affects of this data type change when the code above is executed.

Question 2.

The Single data type in a language complies with the IEEE 754 32-bit Single Precision Floating Point standard.

(a) A variable of type Single currently contains the decimal value 20. The memory location associated with this variable contains the following 32 bits:

01000001101000000000000000000000

Explain how the bit pattern above represents the decimal value 20. Show all working.

(b) The Integer data type in this language is defined as follows:

> **Integer Data Type**
> 32-bit (4-bytes) signed two's complement integer

Compare the Single and Integer data types in terms of their precision, range and efficiency of processing.

Suggested solutions

Question 1.

(a) 'A' contains 180 which we convert into binary:

180	
÷ 2	0
90	
÷ 2	0
45	
÷ 2	1
22	
÷ 2	0
11	
÷ 2	1
5	
÷ 2	1
2	
÷ 2	0
1	
÷ 2	1
0	

'B' contains 9 which we convert into binary:

9	
÷ 2	1
4	
÷ 2	0
2	
÷ 2	0
1	
÷ 2	1
0	

(Check: 8 + 1 = 9 ✓)

(Check: 128 + 32 + 16 + 4 = 180 ✓)

Therefore the content of variable A is represented in memory as 10110100 and the content of variable 'B' is represented in memory as 00001001.

(b)

```
        10100  .
1001 ) 10110100
       1001
         1001
         1001
            000
```

(Check: 16 + 4 = 20 = 180 / 9 ✓)

(c) $A = 1011\ 0100_2 = (8+2+1)\ (4) = B4_{16}$

$B = 0000\ 1001_2 = (0)\ (8+1) = 9_{16}$

$C = 0001\ 0100_2 = (1)\ (4) = 14_{16}$

Therefore the output from line 50 is "B4 divided by 9 equals 14".

(d) There are two possible consequences of the change from unsigned to signed two's complement. Both problems are with the number 180 stored in variable A within the code. 180 is within the range of an unsigned byte (0 to 255) but is outside the range of a signed two's complement byte (-128 to 127).

During execution a runtime error may occur if the programming language detects the now out of range assignment of 180 to the Byte variable A.

The language may allow 180 and let it overflow such that it will be interpreted as a negative number. This means that A (which holds the binary number 1011 0100) will not be interpreted correctly as 180. Because the leftmost bit is a 1, the value would be interpreted instead as a negative number. Using two's complement the decimal value of 1011 0100 = -128 + 32 + 16 + 4 = -76. This will have obvious processing consequences in the code since the intended number is not being used.

Question 2.

(a) Sign = 0

Exponent = 1000 0011

Mantissa = 01000000000000

With a bias of 127, the exponent = (128 + 2 + 1) – 127 = 131 – 127 = 4

With normalisation (add the leading 1), the mantissa = 1.01

Therefore the binary value = $1.01 \times 2^4 = 10100$

Which in decimal = 16 + 4 = 20

(b) Integer values represented as 2's complement signed integers using 4 bytes (32 bits) can take values from -2^{31} to $2^{31} - 1$. Because they are integer values, all whole numbers in this range can be represented exactly as sums of powers of 2, with no need for approximations.

On the other hand, the maximum value that can be represented in a 4 byte floating point number is

= 0 11111110 111111…..1
$= +2^{(254-127)} \times 1.1111111\ldots1$
$\approx 2^{127} \times 2$
$\approx 2^{128}$

2^{128} is clearly significantly larger than the maximum value of $2^{31} - 1$ able to be expressed using the integer representation.

If integer representation is used, only whole numbers can be represented, no fractional values are possible. If floating point representation is used, then it is possible to represent a whole range of fractional values.

However, if floating point representation is used, there are a whole range of numbers that cannot be expressed exactly (such as 9.45). All of these numbers are therefore expressed as an approximation using all 23 binary places available in the mantissa to provide a value very close to that required.

This can give rise to some unexpected results when comparing integer and floating values, where it cannot be assumed that the integer representation is exactly equal to the floating point value, which may be an approximation to the required value as a result of previous calculations.

In terms of processing time required, arithmetic using floating point numbers is nowhere near as straightforward or as efficient as arithmetic using binary integers. In fact floating point arithmetic is performed by a special co-processor chip (or floating point unit) as the mantissa and exponent need to be considered separately and processed differently.

Notes

- In question 1 (a) when using the divide by 2 conversion method the answer is read from top to bottom. Of course, alternative methods of conversion are equally valid.
- The suggested solutions for questions 1 (a) and (b) includes a check to ensure correctness. This is easy to achieve and hence well worth the time it takes, particularly when the question is given in decimal.
- It is important to show all working for these type of questions as often marks will be awarded for correct working and in addition, working often prevents trivial errors being made.

SET 10C

1. The binary addition 00101011 + 10110111 equals:
   (A) 01100011
   (B) 01100010
   (C) 00100010
   (D) 11100010

2. In 8 bit two's complement the addition 11000011 + 10110000 should result in:
   (A) 01110011
   (B) 101110011
   (C) 10111001
   (D) an overflow error.

3. The decimal subtraction 5 - 3 is performed in 8 bit twos complement as:
   (A) 00000101 - 00000011
   (B) 00000101 + 00000011
   (C) 00000101 + 11111101
   (D) 11111011 + 11111101

4. In twos complement subtracting a negative number is performed by:
   (A) Adding the two numbers.
   (B) Subtracting the twos complement of the second number.
   (C) Adding the twos complement of the second number.
   (D) Calculating the twos complement of both numbers and adding.

5. The binary multiplication 1011 × 110 is equal to:
   (A) 1000010
   (B) 100001
   (C) 1111110
   (D) 10111011

6. The binary division 1010101 ÷ 101 is equal to:
   (A) 1001
   (B) 10001
   (C) 101
   (D) 1101

7. Multiplying by 4 in binary can be achieved by:
   (A) shifting one place to the right.
   (B) shifting two places to the right.
   (C) shifting one place to the left.
   (D) shifting two places to the left.

8. Dividing by 8 in binary can be achieved by:
   (A) shifting two places to the right.
   (B) shifting three places to the right.
   (C) shifting two places to the left.
   (D) shifting three places to the left.

9. 1/5 ≈ 1/8 + 1/16 + 1/128 as a consequence the binary division 1100100 by 101 could be accomplished by:
   (A) shifting 1100100 3 places to the right, 1100100 4 places to the right and 1100100 7 places to the right The sum of these three results is the answer..
   (B) shifting 1100100 3 places to the right then 4 places to the right then 7 places to the right.
   (C) Dividing by 2 and then dividing by 8. In other words, shift one to the right then three to the right.
   (D) Subtracting 101 from 1100100 repeatedly until the remainder is less than 101.

10. Using 8 bit twos complement and ignoring overflow errors the decimal addition -100 + -100 results in the decimal answer:
    (A) 72
    (B) -200
    (C) 56
    (D) 0

11. Perform the following binary calculations:

    (a) 1101101 + 110111 (b) 11011 - 1011 (c) 11011 × 1011 (d) 10110011 ÷ 1101

12. When using two's complement to perform calculations programmers must be aware of the range of values that can be represented. Why then would a programmer choose to use integer data types? Wouldn't they be better off using floating-point representations? Discuss.

13. Multiplication can be described as repeated addition e.g. 4 × 5 = 5 + 5 + 5 + 5. Why don't computers use repeated addition to perform binary multiplication?

14. Perform a desk check of the multiplication algorithm in *Fig 10.26* using the binary multiplication 11001 times 1011. Check your result by converting the question and answer into decimal.

15. Perform a desk check of the division algorithm in *Fig 10.29* using the binary division 100010011 divided by 1101. Check your result by converting the question and answer into decimal.

ELECTRONIC CIRCUITS TO PERFORM STANDARD SOFTWARE OPERATIONS

In the previous section, we examined the mathematics involved in the processing of binary numbers. In this section, we examine how these processes are implemented using electronic circuits. For instance, how can an electronic circuit perform binary addition or how can it be used to store binary digits? The answer to these questions will help you to understand the basis of today's digital computers.



Electronic circuit
A circular path along which electrons (electricity) can flow. The circuit must be complete for this to occur.

Logic gates are the basic building blocks for the creation of microprocessors. Used in various combinations logic gates are able to perform all the functions carried out by computers. Essentially, computers are simply a collection of logic gates. The branch of mathematics known as *Boolean algebra* can be used to assist in the efficient design of circuits.

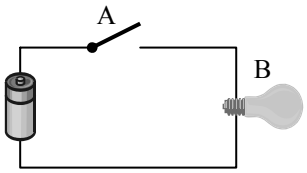


Logic gate
An electronic circuit that transforms inputs into outputs based on strict logical rules. Integrated circuits are made up of combinations of logic gates.

We firstly examine the six basic logic gates and then consider the process of combining these logic gates to perform specific tasks.

LOGIC GATES

There are six basic logic gates that are used in the construction of integrated circuits; AND, OR, NOT, NAND, NOR and XOR. To assist in our discussion of each of these gates, we will use a series of switches together with a battery and light globe. The position of each switch, either off or on, provides the input and the light globe provides the output. For our purposes, a switch in the up or off position is represented by 0 and in the down or on position by 1. Similarly, if the light globe burns this is a 1 and if it doesn't then that is a 0.



A Input	B Output
0	0
1	1

Fig 10.32
A simple circuit. When switch A is up (0) the globe B is out (0). When switch A is down (1) globe B burns (1).

Fig 10.32 describes a simple circuit using just one switch. The table to the right of the circuit is called a truth table. Truth tables are used to summarise all the possible inputs and their resulting outputs for a particular circuit. In this case, an input of 0 results in an output of 0 and an input of 1 results in an output of 1, a useful circuit if you want to turn on and off lights but not particularly useful for processing data. We require circuits that transform the inputs into required outputs.

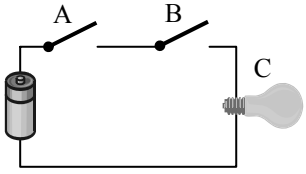
In reality, both the input to and the output from a logic gate is either a high or low voltage representing 1 (true) or 0 (false). This allows the output from one logic gate to be the input for another.



GROUP TASK Discussion
In chapter 5, we learnt about how the fetch-execute cycle is used to implement machine code instructions. Logic gates are the hardware circuits used to perform machine code instructions.
How is this knowledge relevant to software developers? Discuss.

AND gate

This circuit emulates the logical AND operator. If 0 is false and 1 is true then both inputs must be true for the resulting output to be true. The two switches A and B are connected in series; as a result, both switches must be down for the globe to burn.



A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

Fig 10.33
AND gate. Both A and B must be 1 for the light to burn.

The truth table contains two inputs, A and B, as a result there are 4 possible combinations resulting in 4 rows in the table. It is convenient to write down each row in a logical sequence. The combination of columns A and B is written in ascending binary order; namely 00, 01, 10, 11 which is the decimal equivalent of 0, 1, 2, 3.

Each logic gate is represented on circuit diagrams as a symbol. AND gates use a rectangle with a rounded right hand side. For most purposes, two inputs are sufficient, however it is possible to design AND circuits using any number of inputs; 2, 3 or 4 inputs are common. *Fig 10.34* shows the symbol used to represent AND gates. The logical expression A AND B is written in Boolean algebra notation as A·B or just AB.

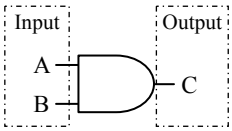
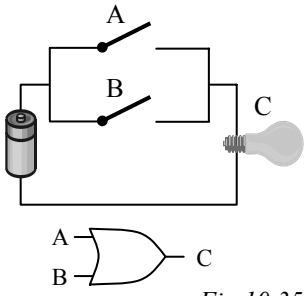


Fig 10.34
The symbol for an AND gate.

OR gate

Either A or B must be true for the output to be true. The two switches are connected in parallel, therefore if either switch is in the down or on position the globe will light. It is possible to have more than two inputs into an OR gate. As long as one or more inputs are true (1) the output will also be true (1). Conversely, if all the inputs are false (0) then the output will be false (0). The logical expression A OR B is written in Boolean algebra notation as A+B.

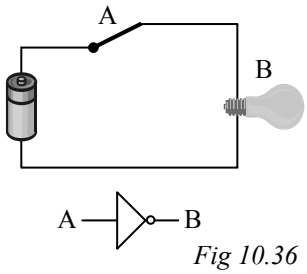


A	B	C
0	0	0
0	1	1
1	0	1
1	1	1

Fig 10.35
OR gate. Either A or B must be 1 for the light to burn.

NOT gate

The NOT gate is used to reverse the input; as a result NOT gates are often called inverters. If the input is false (0) then the output will be true (1) and vice versa. It is only possible to have one input into a NOT gate. Remember that in our representation using switches, battery and globe that a switch in the up position is off (0 or false) and a switch in the down position is on (1 or true). NOT(A) is written in Boolean algebra notation as $\overline{A}$.



A	B
0	1
1	0

Fig 10.36
NOT gate. When switch A is up (0) the globe burns (1). When switch A is down (1) the globe does not burn (0).

The symbol for a NOT gate is a small hollow circle. The triangle preceding the circle is really a buffer. Buffers are used to synchronise timing, often your circuit will need to wait for the next input. Buffers can be used to accomplish this task. NOT gates require buffers to ensure correct timing.

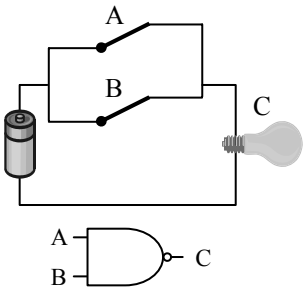
NAND gate

NAND is short for NOT AND, in other words a NAND gate is logically the same as an AND gate followed by a NOT gate. The output from a two input NAND gate is the opposite of that from an AND gate, that is 1110 rather than 0001. The ability to use the output from one logic gate as the input for another provides the ability to combine gates to perform new and varied tasks. The symbol for a NAND gate is the AND symbol followed by the NOT symbol or in Boolean algebra notation as $\overline{AB}$.



Fig 10.37
A NAND gate is the equivalent of an AND gate followed by a NOT gate.

In logic terms, the output from a NAND gate is true if all inputs are not true. This logical explanation holds true regardless of the number of inputs into the NAND gate. Most NAND gates contain 2, 3 or 4 inputs.



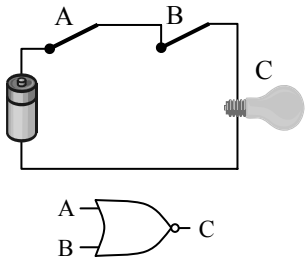
A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

Fig 10.38
NAND gate. All other logic gates can be simulated using NAND gates.

NAND gates are perhaps the most useful logic gate. It is possible to simulate all other logic gates using combinations of NAND gates. As a result, microprocessors could be designed using NAND gates exclusively. Although this is possible, it is not the most efficient method. In the next section, we will attempt to create each of the other logic gates using only NAND gates.

NOR gate

NOR means NOT OR, a NOR gate is equivalent to an OR gate followed by a NOT gate. In Boolean algebra notation NOR is written as $\overline{A+B}$. *Fig 10.39* shows a possible circuit together with the truth table and symbol for the NOR gate. All inputs must be false for the output to be true.

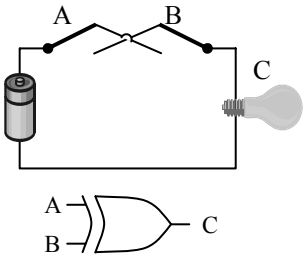


A	B	C
0	0	1
0	1	0
1	0	0
1	1	0

Fig 10.39
NOR gate. Both A and B must be 0 for the light to burn.

XOR gate

XOR is short for exclusive OR. This means that precisely one input must be 1 (true) for the output to be 1 (true). A XOR B is written in Boolean algebra notation as $A \oplus B$. In some literature, XOR is called EOR; the terms are interchangeable. In this text, we will use the term XOR.



A	B	C
0	0	0
0	1	1
1	0	1
1	1	0

Fig 10.40
XOR gate. Exactly one input must be true for the output to be true.

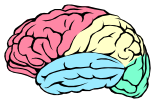


GROUP TASK Discussion
Often staircases have a switch to turn on and off the light at the bottom of the stairs and another switch to turn the same light off and on at the top of the stairs. Which of the above circuits could be used to wire this light. Explain your answer.

CIRCUIT DESIGN

Combinations of the above logic gates can be used to solve new problems. Problems in circuit design are normally presented in the form of inputs and required outputs, essentially a truth table. Computer Aided Design software is available to assist in the design of circuits, in fact it would be impossible for today's microprocessors to have been created without the use of computer aided circuit design tools!

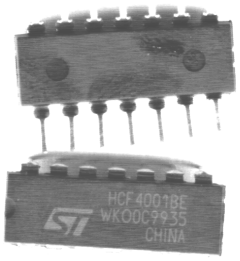
The process of developing new circuit designs is cyclical in nature. As is the case with software solutions, there are often many ways to solve the same problem. The aim is to find the most efficient and elegant solution. This often involves a process of evolutionary refinement. In this course, we need only consider the logical level of electronic circuit design. Electrical engineers need to consider many other factors when solving real world problems.



Consider the following:

Imagine you are designing a circuit for an electronics project you are working on at home. You visit your local electronics store and find that they only have integrated circuits (ICs) containing NAND gates. *Fig 10.41* shows the pin assignments for one of the available chips. These chips contain 4 NAND gates e.g. the result of inputs at pin 1A and 1B is output at pin 1C.

For your project you require AND, OR, NOT, NOR, XOR and NAND gates. How can you simulate each of these gates using only NAND integrated circuits?



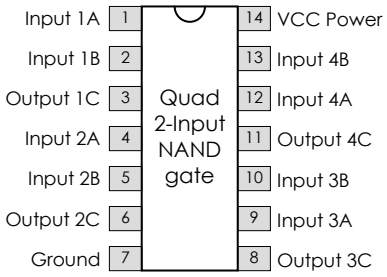


Fig 10.41
74HC series CMOS NAND integrated circuit. Contains 4 NAND gates.

NOT gate

Examining the truth table for NAND indicates that when both input A and B are the same i.e. both 0s or both 1s we effectively have a NOT gate. Logically we perform (A NAND A). Therefore, if we connect input A and B together to form a single input the result is a simulated NOT gate e.g. connect pins 1 and 2 on our IC. *Fig 10.42* shows a circuit diagram achieving this purpose.

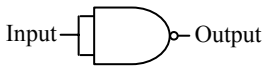


Fig 10.42
NOT gate produced using a NAND gate.

AND gate

The truth table for a NAND gate is the opposite of that for an AND gate. Therefore, if we reverse a NAND gate we should arrive back at an AND result. In essence we are performing NOT(A NAND B) which is accomplished using a NAND gate followed by a NOT gate. Using our IC we could connect pins 3, 4 and 5 together, our inputs would then be at pins 1 and 2 with our AND result output at pin 6. *Fig 10.43* shows a circuit design together with the possible connections required using our IC.

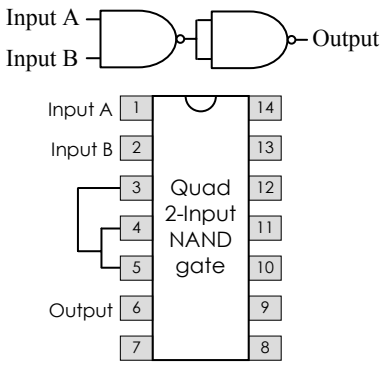


Fig 10.43
AND gate produced using 2 NAND gates.

OR gate

This gate requires a little more thought. Let us examine the truth tables for both the NAND and OR gates side by side (*see Fig 10.44*). Inputs of 01 and 10 result in the same outputs of 1 from both NAND and OR gates, however the outputs from 00 and 11 are reversed. If we can invert the inputs before they reach the NAND gate, we may get the result we desire.

A	B	NAND	OR
0	0	1	0
0	1	1	1
1	0	1	1
1	1	0	1

Fig 10.44
Truth table for NAND and OR side-by-side.

In logical terms we perform (NOT(A)) NAND (NOT(B)). As we don't have any NOT gates we substitute each NOT for our NAND equivalent, hence our logical statement becomes (A NAND A) NAND (B NAND B). We require three NAND gates to produce a single OR gate. *Fig 10.45* shows the circuit diagram and pin connections using our NAND IC.

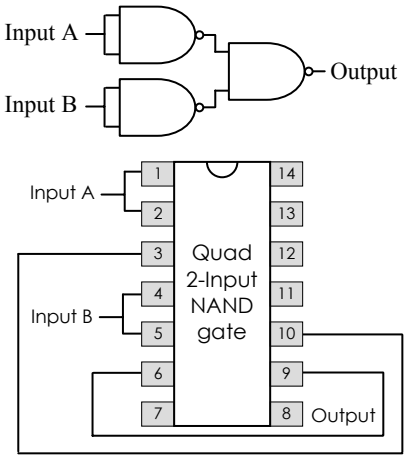


Fig 10.45
OR gate produced using 3 NAND gates.

XOR gate

This gate is rather more difficult. Let us first examine the truth table for the XOR gate, the outputs are 0110. Similar in many ways to an OR gate except 1 and 1 results in an output of 0 rather than 1. If we consider the last sentence and attempt to write it as a logical statement we get something like: A or B but not A and B. This can be written as (A OR B) AND NOT(A AND B). Now NOT(A AND B) is the opposite of an AND gate, which fortunately is a NAND gate; our statement becomes (A OR B) AND (A NAND B). *Fig 10.46* shows our progress as a circuit diagram.

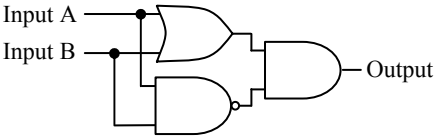


Fig 10.46
XOR gate produced using 3 different gates.

We could replace the OR and AND gate in this solution with the NAND representations we developed above, this results in the circuit design shown in *Fig 10.47*. Unfortunately, this solution requires 6 NAND gates and each of our ICs contains only 4 NAND gates. It would be convenient and more efficient to find a solution using just 4 NAND gates.

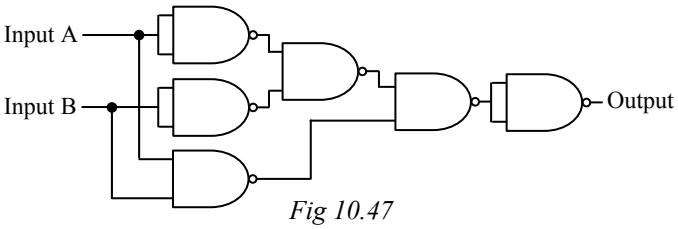


Fig 10.47
XOR gate produced using 6 NAND gates.

Boolean algebra provides a mathematical technique for simplifying and/or rearranging circuit designs such as this. The following pages describe the basics of Boolean algebra and how it can be used to optimise the design of circuits.



GROUP TASK Research
Research using the internet (or otherwise) the different chips available in the 74xxxx series. Determine the function of each pin for each of the chips you find.

BOOLEAN ALGEBRA

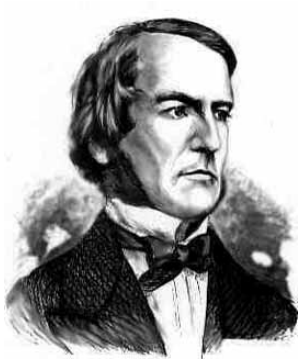


Fig 10.48
George Boole (1815-64)

Boolean algebra was first developed by the English mathematician George Boole around 1850. Coincidentally, this was about the same time Charles Babbage was working on his Analytical Engine, the forefather of today's computers. Boolean algebra is used to describe logical propositions where the outcome is either true or false. The application of Boolean algebra to circuit design has enabled the design of today's efficient circuits containing many millions of logic gates.

In Boolean algebra notation, variables can hold values of 0 (false) or 1 (true). There are three operations NOT, OR and AND. NOT A is denoted as $\overline{A}$, A OR B is denoted A+B and A AND B is denoted $A \cdot B$ or just AB. A further notation has been added for XOR, A XOR B is denoted $A \oplus B$.

Boole, along with one of his contemporaries, Augustus DeMorgan developed and proved the following laws of Boolean algebra:

1a. A+B=B+A
1b. AB=BA
2a. (A+B)+C=A+(B+C)
2b. (AB)C=A(BC)
3a. A(B+C)=AB+AC
3b. A+(BC)=(A+B)(A+C)
4a. A+A=A
4b. AA=A
5a. 0+A=A
5b. 0A=0
6a. 1+A=1
6b. 1A=A
7a. $\overline{A}$ +A=1
7b. $\overline{A}$ A=0
8a. A+$\overline{A}$ B=A+B
8b. A($\overline{A}$ +B)=AB

DeMorgan's Theorem

9a. $(\overline{A+B})=\overline{A}\ \overline{B}$
9b. $(\overline{AB})=\overline{A}+\overline{B}$

Let us use these laws to help simplify our XOR problem into a series of NAND gates. At this stage it is not necessary to be able to reproduce such working rather it is sufficient to be able to follow each step in the working based on the above laws of Boolean algebra. Initially we described XOR as A OR B AND NOT(A AND B):

$$A \oplus B = (A+B)\overline{AB} \quad \textit{(definition of XOR gate)}$$
$$= \overline{AB}(A+B) \quad \textit{(using law 1b.)}$$
$$= \overline{AB}A + \overline{AB}B \quad \textit{(using law 3a.)}$$
$$= A\overline{AB} + B\overline{AB} \quad \textit{(using law 1b.)}$$
$$= \overline{\overline{A\overline{AB}} + \overline{B\overline{AB}}} \quad \textit{(applying not twice).}$$
$$= \overline{\overline{A\overline{AB}}}\ \overline{\overline{B\overline{AB}}} \quad \textit{(using law 9a.)}$$

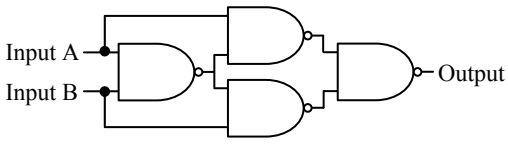


Fig 10.49
XOR gate using only 4 NAND gates.

Implementing this Boolean expression as a circuit diagram requires 4 NAND logic gates (see *Fig 10.49*). As a result, we are able to use a single IC; remember each of our ICs contains 4 NAND gates. *Fig 10.50* shows the wiring for our IC.

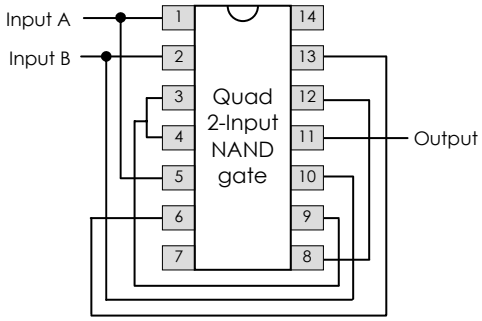


Fig 10.50
XOR gate created using an IC containing 4 NAND gates.

Let us examine the above laws of Boolean algebra in more detail. The first thing we notice is that many of the laws are the same as regular algebra. In particular laws 1a, 1b, 2a, 2b, 3a, 5a, 5b and 6b. This leaves the following set of new laws to learn:

3b. A+(BC)=(A+B)(A+C)
4a. A+A=A
4b. AA=A
6a. 1+A=1
7a. $\overline{A}$ +A=1
7b. $\overline{A}$ A=0
8a. A+ $\overline{A}$ B=A+B
8b. A($\overline{A}$ +B)=AB
9a. ($\overline{A+B}$)= $\overline{A}$ $\overline{B}$
9b. ($\overline{AB}$)= $\overline{A}$ + $\overline{B}$

Let us briefly run through each of these laws commencing with the simpler laws first.

- 4a. A+A=A. In words this law means "A or A will equal A regardless of the value of A". *Fig 10.51* shows the truth table which confirms that when A=0 then A+A also equals 0, and when A is 1 then A+A also equals 1 based on our earlier OR gate description. In practice this rule is often extended in a variety of ways such as, AB + AB = AB, A+A+A+A = A and even in reverse when an extra term is needed such as replacing AB with AB + AB within an expression.

A	A	A+A
0	0	0
1	1	1

Fig 10.51
Law 4a truth table

- 4b. AA=A. This law means "A and A always equals A regardless of the value of A". The truth table in *Fig 10.52* confirms this to be true based on our previous AND gate definition and truth table. Can also be extended so it is also true to say that AAAAA = A or in combination with rule 1b it is true that ABBAABAB = AAAABBBB = AB.

A	A	AA
0	0	0
1	1	1

Fig 10.52
Law 4b truth table

- 6a. 1+A=1. This means when 1 is 'OR'ed with anything the result is always 1. This is a very useful rule as it often allows a whole string of complex terms to be reduced to 1. For example, the expression AB+BD+AD+1 = (AB+BD+AD)+1 = 1.

1	A	1+A
1	0	1
1	1	1

Fig 10.53
Law 6a truth table

- 7a. $\overline{A}$ +A=1. If two opposite Boolean values are 'OR'ed the result is always 1 (true).
- 7b. $\overline{A}$ A=0. If two opposite Boolean values are 'AND'ed the result is always 0 (false).

$\overline{A}$	A	$\overline{A}$ + A	$\overline{A}$A
0	1	1	0
1	0	1	0

Fig 10.54
Law 7a and 7b truth table

- 3b. A+(BC)=(A+B)(A+C). This is a very useful law and it is not immediately obvious that it is true. The following mathematical proof shows the rule to be true however a truth table could also be used to confirm the law.

RHS = (A+B)(A+C)
= AA+AC+BA+BC (expanding using law 3a)
= A+AC+AB+BC (as AA=A law 4b)
= A(1+C+B)+BC (factorising using law 3a)
= A(1)+BC (using law 6a)
= A+BC (as 1A=A law 6b)
= LHS

- 8a. $A+\overline{A}B=A+B$. The following mathematical proof and the truth table in *Fig 10.55* shows the law to be true.

RHS = A+B
= A+B(1) (as 1B=B law 6b)
= A+B(A+$\overline{A}$) (using law 7a)
= A+AB+$\overline{A}$B (expanding using law 3a)
= A(1+B)+$\overline{A}$B (factoring using law 3a)
= A(1)+$\overline{A}$B (using law 6a)
= A+$\overline{A}$B (using law 6b)
= LHS

A	B	$\overline{A}$	$\overline{A}B$	$A+\overline{A}B$	$A+B$
0	0	1	0	0	0
0	1	1	1	1	1
1	0	0	0	1	1
1	1	0	0	1	1

Fig 10.55
Law 8a truth table

- 8b. $A(\overline{A}+B)=AB$. The following mathematical proof and the truth table in *Fig 10.56* shows the law to be true.

LHS = A($\overline{A}$+B)
= A$\overline{A}$+AB (expanding using law 3a)
= 0+AB (using law 7b)
= AB (using law 5a)
= RHS

A	B	$\overline{A}$	$\overline{A}+B$	$A(\overline{A}+B)$	AB
0	0	1	1	0	0
0	1	1	1	0	0
1	0	0	0	0	0
1	1	0	1	1	1

Fig 10.56
Law 8b truth table

- 9a. $(\overline{A+B})=\overline{A}\ \overline{B}$ and 9b. $(\overline{AB})=\overline{A}+\overline{B}$. These are DeMorgan's theorems and are the basis for the "Sum of Products" and "Products of Sums" techniques we examine in the next section. In words theorem 9a means the reverse of 'OR'ing is the same as reversing each value and then 'AND'ing them. And 9b means the reverse of 'AND'ing is reversing then 'OR'ing. These theorems are useful when you wish to convert OR gates to AND gates or vice versa. The following truth tables in *Fig 10.57* shows both theorems to be true.

A	B	$\overline{A}$	$\overline{B}$	$\overline{A+B}$	$\overline{A}\,\overline{B}$	$\overline{AB}$	$\overline{A}+\overline{B}$
0	0	1	1	1	1	1	1
0	1	1	0	0	0	1	1
1	0	0	1	0	0	1	1
1	1	0	0	0	0	0	0

Fig 10.57
DeMorgan's Theorem 9a and 9b truth tables



GROUP TASK Activity
We neglected to construct the truth table to confirm law 3b. Construct a suitable truth table.



GROUP TASK Activity
Each of the Boolean algebra laws can also be viewed as equivalent circuits. In law 8a the circuit A+B is logically the same as the circuit A+$\overline{A}$B. Draw both circuit diagrams for both these circuits.



GROUP TASK Research
Boolean algebra is not just used when designing circuits it has a wide variety of other applications. Research and describe some of these other applications of Boolean algebra.

CONVERSION BETWEEN TRUTH TABLES, BOOLEAN EXPRESSIONS AND CIRCUIT DIAGRAMS

Often problems present themselves in words which can readily be expressed as a truth table. However to convert such truth tables into a circuit design is often a difficult task. However if we can convert the truth table to an equivalent Boolean expression then we can not only simplify the expression using Boolean algebra but it is also a much simpler task to construct our circuit diagram from the final Boolean expression.

There are two strategies for deriving a Boolean equation from any truth table:

- Sum of Products (SoP) which uses all the truth table rows with a 1 output.
- Product of Sums (PoS) which uses all the truth table rows with a 0 output BUT with each input reversed.

Usually the best strategy to use is the one with the least number of rows. So if there are less 1s than 0s in the output column then use SoP and if there are less 0s than 1s then use PoS.

A trivial example to introduce the SoP and PoS strategies. Say we wish to develop an efficient circuit to implement the truth table in *Fig 10.58*. In this case there are two inputs, namely A and B and a single output X. If we inspect the truth table we can probably see that that X is always equal to the opposite of A. In Boolean algebra this is written $X=\overline{A}$. This is the solution we are hoping to determine using SoP and also PoS. Remember this is a trivial example; for more difficult cases it will not be possible to see the solution by simply inspecting the truth table.

A	B	X
0	0	1
0	1	1
1	0	0
1	1	0

Fig 10.58
Sample truth table

Check the output column to see whether there are less 1s or less 0s. In this case the output or X column contains an equal number of 1s and 0s so it doesn't really matter whether we choose SoP or PoS as our strategy (although SoP is often simpler in such cases).

Using Sum of Products (SoP) we use only the rows with an output of 1 – the first two rows in this case. The first row is A=0 and B=0 which written as a product gives us $\overline{A}\ \overline{B}$. The second row is A=0 and B=1 so our product from this row is $\overline{A}B$. The sum of these products is therefore $\overline{A}\ \overline{B}+\overline{A}B$. So are final Boolean equation is $X=\overline{A}\ \overline{B}+\overline{A}B$ which we can simplify using Boolean algebra as follows:

$$\begin{aligned} X &= \overline{A}\ \overline{B}+\overline{A}B \\ &= \overline{A}(\overline{B}+B) \\ &= \overline{A}(1) \\ &= \overline{A} \end{aligned}$$

Now let us try using Products of Sums (PoS). This time we only use the rows with an output of 0 which is the last two rows in *Fig 10.58*. The third row is A=1 and B=0 but we need to reverse these values to form our first sum. Therefore the sum from row three of the truth table is $\overline{A}+B$. The fourth row is A=1 and B=1. Again reverse the values and add to form the sum $\overline{A}+\overline{B}$. The product of these sums is therefore $(\overline{A}+B)(\overline{A}+\overline{B})$ and the final Boolean equation is $X=(\overline{A}+B)(\overline{A}+\overline{B})$. Which using Boolean algebra simplifies to $X=\overline{A}$ as expected. Note that in this trivial example the input B is not required at all and the circuit diagram is simply a single NOT gate.



GROUP TASK Activity

Use Boolean algebra to show that the equation $X=(\overline{A}+B)(\overline{A}+\overline{B})$ does indeed simplify down to $X=\overline{A}$ as expected.

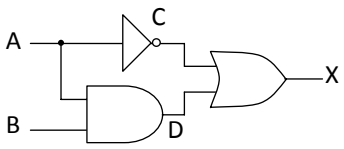


HSC style questions:

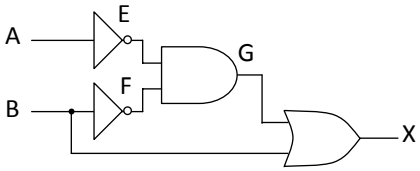
Question 1.

A student has designed the following two circuits.

Circuit 1.



Circuit 2.



(a) Complete truth tables to prove both the above circuits produce the same output. Include each letter on the circuits as a column in the truth tables.

(b) Design a simpler circuit that produces the same output as the above circuits.

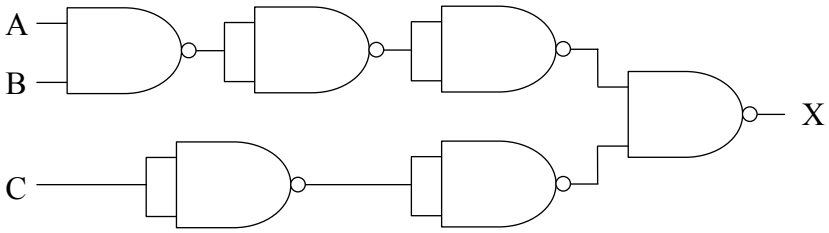
Question 2.

A	B	C	Output
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Create a circuit diagram based on the above truth table.

Question 3.

The following circuit uses six NAND gates.



(a) Construct a truth table for the above circuit.

(b) Design a new circuit using only AND, OR and/or NOT gates that efficiently produces the same set of outputs.

Suggested solutions

Question 1.

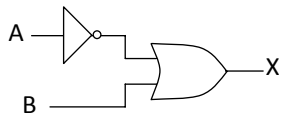
(a)

Circuit 1.				
A	B	C	D	X
0	0	1	0	1
0	1	1	0	1
1	0	0	0	0
1	1	0	1	1

Circuit 2.					
A	B	E	F	G	X
0	0	1	1	1	1
0	1	1	0	0	1
1	0	0	1	0	0
1	1	0	0	0	1

The X columns are the same therefore both circuits are logically the same as they output the same values for the equivalent inputs.

(b) Using Boolean algebra and PoS (Product of Sums) with the 3rd line of the truth table we get $X = \overline{A} + B$. This circuit requires just 2 logic gates and is drawn as

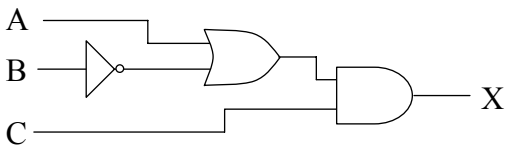


Question 2.

As there are less rows with an output of 1 we will use SoP (Sum of Products) with these three rows.

$$\begin{aligned}\text{Output} &= \overline{A}\,\overline{B}\,C + A\,\overline{B}\,C + ABC \\ &= C(\overline{A}\,\overline{B} + A\,\overline{B} + AB) \\ &= C(\overline{A}\,\overline{B} + A(\overline{B} + B)) \\ &= C(\overline{A}\,\overline{B} + A(1)) \\ &= C(\overline{A}\,\overline{B} + A) \\ &= C(\overline{B} + A)\end{aligned}$$

Which is equivalent to the following circuit diagram.



Question 3.

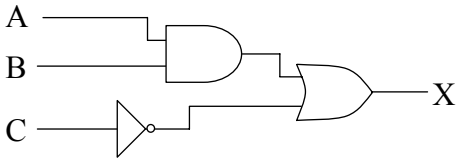
(a)

A	B	C	$\overline{AB}$	X
0	0	0	1	1
0	0	1	1	0
0	1	0	1	1
0	1	1	1	0
1	0	0	1	1
1	0	1	1	0
1	1	0	0	1
1	1	1	0	1

(b) As there are less rows with a 0 output we use PoS (Product of Sums). the rows have inputs of 001, 011 and 101 which we reverse to form our PoS equation.

$$
\begin{aligned}
X &= (A+B+\overline{C})\,(A+\overline{B}+\overline{C})\,(\overline{A}+B+\overline{C}) \\
&= (A+B+\overline{C})\,(A+\overline{B}+\overline{C})\,(\overline{A}+B+\overline{C}) \\
&= (A+B+\overline{C})\,(0+AB+A\overline{C}+\overline{A}\ \overline{B}+0+\overline{B}\ \overline{C}+\overline{A}\ \overline{C}+B\overline{C}+\overline{C}) \\
&= (A+B+\overline{C})\,(AB+A\overline{C}+\overline{A}\ \overline{B}+\overline{B}\ \overline{C}+\overline{A}\ \overline{C}+B\overline{C}+\overline{C}) \\
&= (A+B+\overline{C})\,(AB+\overline{A}\ \overline{B}+A\overline{C}+\overline{B}\ \overline{C}+\overline{A}\ \overline{C}+B\overline{C}+\overline{C}) \\
&= (A+B+\overline{C})\,(AB+\overline{A}\ \overline{B}+\overline{C}\,(A+\overline{B}+\overline{A}+B+1)) \\
&= (A+B+\overline{C})\,(AB+\overline{A}\ \overline{B}+\overline{C}) \\
&= AAB+A\overline{A}\ \overline{B}+A\overline{C}+BAB+B\overline{A}\ \overline{B}+B\overline{C}+\overline{C}\,AB+\overline{C}\ \overline{A}\ \overline{B}+\overline{C}\ \overline{C} \\
&= AB+0\overline{B}+A\overline{C}+AB+0\overline{A}+B\overline{C}+\overline{C}\,AB+\overline{C}\ \overline{A}\ \overline{B}+\overline{C} \\
&= AB+AB+A\overline{C}+B\overline{C}+\overline{C}\,AB+\overline{C}\ \overline{A}\ \overline{B}+\overline{C} \\
&= AB+\overline{C}\,(A+B+AB+\overline{A}\ \overline{B}+1) \\
&= AB+\overline{C}
\end{aligned}
$$

Which is equivalent to the following circuit design.



Notes

- In question 1 (a) the C, D and the E, F, G columns are required for full marks. As the question indicates the outputs will be the same then these extra inputs are what shows the processing performed within the circuit.
- The 6 NAND circuit diagram in question 3 is not as complex as it first looks. A NAND gate with a single input is logically a NOT gate. In this circuit there are two places where 2 of these NOT gates follow each other which means they cancel each other out and can be ignored. This greatly reduces the work to be done constructing the truth table in part (a).
- Question 3 (b) is definitely a difficult problem to solve from the truth table using Boolean algebra alone. It may have been simpler to attempt to find a solution by inspecting the truth table. Looking at the C and X columns we can see that X is pretty much the opposite of C. The only exception is the last row where A, B and C are 1. Using this observation one could conceivably determine the answer $X= AB+\overline{C}$ directly.
- Another alternative for question 3 (b) is to write the Boolean equation based on the original 6 NAND circuit diagram and then simplify using Boolean algoebra.

$$
\begin{aligned}
X &= \overline{\overline{\overline{AB}\,\overline{C}}} \\
&= \overline{\overline{AB}\,\overline{C}} \quad \textit{Removing the two sets of double NOTs} \\
&= AB+\overline{C} \quad \textit{Using 9b (DeMorgan's theorem)}
\end{aligned}
$$

- To simplify writing Boolean expressions using word processors it is common in some references to use an apostrophe as the symbol for NOT. That is, A' is the same as $\overline{A}$, (A+B)' is the same as $\overline{A+B}$ and (AB)' is the same as $\overline{AB}$.



GROUP TASK Activity
Use SoP to answer questions 1(b) and 3(b) above. And use PoS to answer question 2. Check you get the same answers as above. Comment on the relative difficulty compared to the suggested solutions above.



GROUP TASK Activity
Draw a truth table with three input columns A,B, C and an output column X. Write random values in the X column. Use both SoP and PoS to help create a circuit diagram.

SPECIALITY CIRCUITS

There are a number of circuits that are in such common usage they are available packaged within ICs and are an integral part of modern CPUs. These circuits include adders, flip-flops and shift registers. In this section, we examine the circuit designs behind each of these specialty circuits.

Binary adders

Addition of binary digits is a crucial operation for most software products hence it makes sense to provide standard circuits to carry out this task. Let us examine the process of designing a circuit to perform 8 bit binary addition.

We commence by considering the truth table required for the addition of two bits (see *Fig 10.59*). If we can develop a circuit for two bit addition perhaps we can reproduce our design to extend the solution to 8 bits.

Input		Output	
A	B	Carry	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Fig 10.59
Truth table for the addition of two bits.

The carry column in the truth table is identical to an AND gate truth table and the sum column is identical to the XOR truth table. Our circuit design will therefore require an AND and an XOR gate. Using Boolean algebra, Sum= $A \oplus B$ and Carry = AB. The completed circuit is shown in *Fig 10.60*.

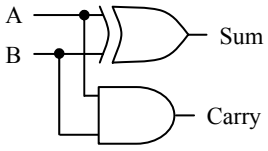


Fig 10.60
Binary half adder

This circuit is great if we just wish to add two bits, however we wish to extend our addition circuit to add 8 bits. There is a problem with this circuit, as it does not take account of possible carries from the previous addition. In fact, this circuit does only half the job and is therefore called a 'half adder'.



GROUP TASK Activity
There are a number of methods of constructing half adder circuits. Can you come up with two unique designs that are different to that shown in *Fig 10.60.*



GROUP TASK Discussion
It is not necessary for microprocessors to contain subtraction circuitry. Why do you think this is the case? Discuss.

Let us now design a circuit that takes account of a previous carry. This circuit will need 3 inputs; A, B and Carry In, together with outputs for the Sum and Carry Out. As there are three inputs to this circuit, our truth table will require 2^3=8 rows (see *Fig 10.61*). The binary addition of the inputs results in the outputs shown in the table.

Input			Output	
Carry In	A	B	Carry Out	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Fig 10.61
Truth table for the addition of two bits with carry in and carry out.

Examining this truth table seems at first glance, to indicate that the Carry Out behaves like an AND gate when Carry In is 0 and like an OR gate when the Carry In is 1. Similarly, the Sum behaves like an XOR gate and a NOT(XOR) gate. On further examination, we find that the sum can be obtained by performing Carry In XOR (A XOR B) and the Carry Out is 1 if two or three of the inputs are 1s.

Comparing these observations with the half adder circuit indicates we can use two half adders to create a full adder circuit. If either of the half adders produces a carry then there needs to be an output carry. As a result Carry Out is the result of an OR gate between the carry outputs from the two half adders. *Fig 10.62* shows a full adder created in this way.

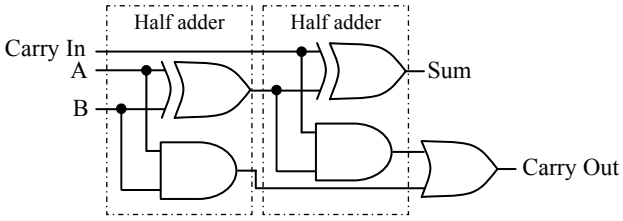


Fig 10.62
A full adder combines two half adders.

We now have a circuit that successfully adds two binary digits allowing for a carry from a previous addition and sending a carry to the next addition. How can we use this circuit to add 8 bits? We need merely to daisy chain these full adders together. The Carry Out from one full adder becoming the Carry In for the next. Rather than redraw all the detail of each full adder we can draw it as a black box with just inputs and outputs shown. *Fig 10.63* shows the final circuit design for adding two 8 bit numbers. The first number is input into the circuit via the eight A inputs and the second via the eight B inputs. The result is obtained from the eight Sum outputs.

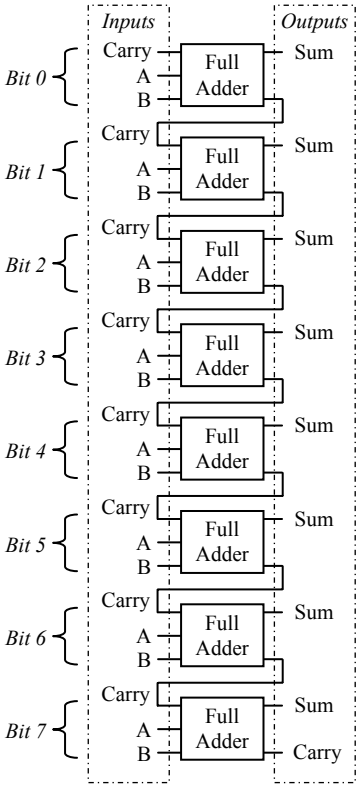


Fig 10.63
An 8 bit adder created using 8 full adder circuits.



GROUP TASK Activity
Draw up a truth table, including intermediate steps for the full adder shown in *Fig 10.62*. Ensure the circuit works as intended.



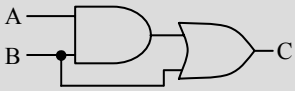
GROUP TASK Discussion
How could the 8 bit adder in *Fig 10.63* be used to create a 16 bit or 32 bit adder? What is the significance of the final carry?

SET 10D

1. If inputs of 01 or 10 both result in an output of 1 then we must be using a:
   (A) AND, OR or NAND gate.
   (B) OR, NAND or NOR gate.
   (C) XOR, OR or NAND gate.
   (D) AND, NOR or XOR gate.

2. Which of the following is true of an AND gate?
   (A) Either input must be true for the output to be true.
   (B) If any of the inputs are true then the output will be true.
   (C) All inputs must be true for the output to be true.
   (D) If the inputs are all false then the output will be true.

Consider the following circuit diagram when answering questions 3, 4 and 5.



3. A truth table with inputs for A and B of 00, 01, 10 and 11 for the above circuit diagram would have the following outputs:
   (A) 0001
   (B) 0111
   (C) 0101
   (D) 1010

4. Which of the following correctly describes the above circuit diagram:
   (A) AB (A+B)
   (B) (A+B) B
   (C) AB + (A+B)
   (D) AB + B

5. Changing the AND gate into a NAND gate would cause the outputs from the above circuit to be:
   (A) 1111
   (B) 0000
   (C) 0101
   (D) 1010

5. Examine the truth table that follows:

A	B	C
0	0	1
0	1	1
1	0	0
1	1	0

   If A and B are inputs and C is an output then the best description of this truth table is:
   (A) A or B but not A and A.
   (B) not A.
   (C) not A and B, or A or B.
   (D) not A and not B, or not A and B

7. The equivalent of A NOR B is:
   (A) NOT(A) AND NOT(B)
   (B) NOT(A AND B)
   (C) A NAND B
   (D) NOT(A) NAND NOT(B)

8. In Boolean algebra notation NOT(A) XOR B would be written:
   (A) $\overline{A + B}$
   (B) $\overline{A \oplus B}$
   (C) $\overline{A} \oplus B$
   (D) $\overline{A} + B$

9. Which of the following is FALSE?
   (A) A full adder has 3 inputs and 2 outputs.
   (B) Full adders contain 2 half adders.
   (C) A half adder is not able to add 2 binary digits correctly.
   (D) 16 half adders are required to create an 8 bit adder.

10. Circuit design can be described as a cyclical process, this is because:
    (A) the outputs from one gate can provide the input for other gates.
    (B) continual refinement of initial designs involves revisiting each step of the design process.
    (C) when evaluating a design using truth tables we need to repeatedly examine each logic gate.
    (D) the inputs into each logic gate can change state in a circular fashion.

11. Simplify each of the following Boolean expressions and draw a circuit diagram.

    (a) $AB + A\overline{B} + \overline{A}B$

    (b) $\overline{A}B + \overline{A}\overline{B} + \overline{B}$

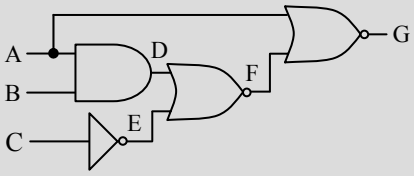
    (c) $B(A + C) + A\overline{B} + B\overline{C} + C$

    (d) $(C + \overline{A}C + \overline{B}C)(A + \overline{A}BC + B)$

12. (a) A warning tune plays if you leave your keys in the ignition of your car or you leave your lights on with the engine off. Either the keys are in the ignition or they are not, similarly either the lights are on or they are not or the engine is running or it is not. Make up a truth table to describe this scenario, the output being the tune playing or not playing. Create a circuit diagram using logic gates for this situation.

    (b) Imagine you require an OR gate but only have an IC containing NOR gates. How can you construct an OR gate using only NOR gates? Draw a circuit diagram to explain your answer.

13. Consider the circuit diagram below.



Input			Output			
A	B	C	D	E	F	G
0	0	0				
0	0	1				
0	1	0				
0	1	1				
1	0	0				
1	0	1				
1	1	0				
1	1	1				

    (a) Copy and complete the above truth table based on the circuit diagram.

    (b) Design a simpler circuit diagram using a minimum number of logic gates.

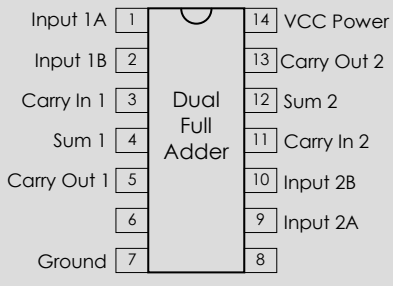
14. Integrated circuits are available that contain 2 full adder circuits. The diagram at right shows the pin assignments for one such IC. Your task is to create an eight bit adder using these ICs.

    Draw four ICs and then show all the connections necessary to form an 8-bit adder.

15. Identify the binary inputs and outputs for each of the following scenarios. Create a truth table using these inputs and outputs. Use your truth table to construct a circuit diagram.

    (a) In hotel rooms, there are two switches; one marked ‘Don’t disturb’ and another marked ‘Please make up my room’. These switches turn on small lights outside the room. If a guest turns both switches on then only the "don’t disturb" light burns.

    (b) A small disposable camera has an inbuilt flash that must be turned on before taking pictures in low light. A light sensor determines if sufficient light is available to take a picture without the flash. The camera will not take a picture if the light is low and the flash is off.



    (c) A virtual pet requires food, water and affection. The pet can be in one of four states; 0 dead, 1 poor, 2 OK and 3 great. If the pet has all three of its requirements, it will be great. If it has no water, it dies. If it just has water, it will be in poor health. If it has two of its requirements then it will be OK.

Flip-flops as memory stores

Flip-flops are bistable circuits used to store binary digits. Bistable means the circuit is able to maintain or remember two different states, representing the binary digits zero and one. Flip-flops are the basis of most temporary memory within computers; this includes registers within the CPU and some types of RAM chips.



Bistable
A stable device is able to hold or maintain a particular state. A bistable device can maintain two possible states.

There are various designs of flip-flop, each with its own advantages and disadvantages for particular applications. All flip-flops have three basic components included within their design, one or more latch components, a clocking component and an edge trigger component. The latch components are the circuits that actually store the data, the remaining two components are designed to control the timing and ensure the integrity of the flip-flop.

Let us consider the design of a typical flip-flop by developing each of these three components:

Latch component

A latch is a circuit that is able to store a binary digit. In the same way that a latch is used to lock a gate, a latch circuit is used to lock a binary digit within a circuit. For a particular state, either 0 or 1, to be maintained within a circuit requires feedback. Feedback is provided by connecting the outputs of two logic gates to each other's inputs.

A simple latch could be constructed using two not gates or inverters; *Fig 10.64* depicts such a circuit. This circuit is able to maintain two states. The first state occurs when Q is 1. If this occurs then the feedback line to the bottom inverter causes $\overline{Q}$ to be 0. As a consequence, the feedback line to the top inverter will cause Q to retain a value of 1, and so on. The second state occurs when Q is 0, in this case Q will maintain a value of zero and $\overline{Q}$ a value of 1. It is traditional to name the outputs from latches and flip-flops Q and $\overline{Q}$. $\overline{Q}$ should always be the opposite of Q.

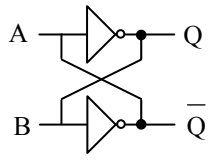


Fig 10.64
A simple latch constructed using two inverters.

The problem with this NOT based latch is that unpredictable results occur if the inputs are altered. To correct this problem we can use one of the two input logic gates. It turns out that either NOR or NAND gates are the best solution. The inputs to these latches are named R and S, meaning reset and set, as a result these latches are named RS NOR and RS NAND latches (see *Fig 10.65*). Some references refer to these circuits as flip-flops, this is not strictly correct as flip-flops require further circuitry to control timing and edge triggering.

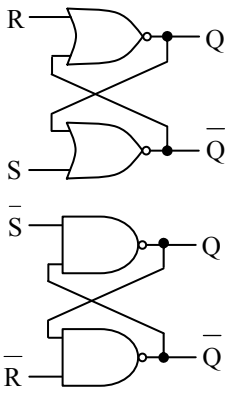


Fig 10.65
RS-NOR latch and RS-NAND latch circuits.

Let us examine the operation of RS latches in more detail. Firstly, the normal holding state for both R and S is 0. To alter the latch to store a 1 requires a 1 to be sent to the set (S) input, similarly a 1 at the reset (R) input will store a 0 in the latch. Notice that our RS-NAND latch circuit has inverted inputs, when working with this RS latch it is necessary to invert the R and S values before calculating the outputs. Both the RS-NOR and RS-NAND latches have the same resultant truth table (see *Fig 10.66*).

Input		Output		
S	R	Q	$\overline{Q}$	
0	0	Last Q	Last $\overline{Q}$	*Hold*
0	1	0	1	*Reset*
1	0	1	0	*Set*
1	1	X	X	*Illegal*

Fig 10.66
Truth table for an RS latch.

There are still problems with these latches. The first being how do we control when a latch is allowed to change state? We need to control this to allow synchronisation of multiple latches. This problem is solved by adding a clock component to our circuit. The next section describes this process.



GROUP TASK Discussion
Work through the operation of each of the RS latches. Ensure you understand how each of these latches produces the truth table shown in *Fig 10.66*. Discuss your findings.

Clocking component

Let us work with our RS-NAND latch, although we could equally have worked with the RS-NOR latch. When a number of latches are connected together, it is vital that we can force all latches to change (or not change) at the same time. By connecting a clock to our circuit, we can achieve this aim. Essentially, a clock is an extra input that changes state from 0 to 1 then back to 0 when we wish to make changes to the attached latches. When the clock input is at 1 then the latch is permitted to change state. When the clock input is at 0 no change to the latches can occur.

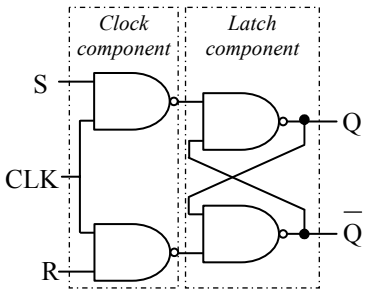


Fig 10.67
Clocked RS-NAND latch circuit.

The advantage of having chosen the RS-NAND latch rather than the RS-NOR latch is that the clock circuitry reverses the inputs, hence we are now working with a hold state of 00 (the original RS-NAND latch required inverted inputs, see *Fig 10.65*). This is preferable as 0 really means low or no voltage, whereas 1 means high voltage.

Two problems remain, what happens if the S or R inputs are both 1 when the clock (CLK) is 1. Also it is possible for S and R to change multiple times whilst CLK is 1.

Input			Output		
S	R	CLK	Q	$\overline{Q}$	
0	0	1	Last Q	Last $\overline{Q}$	*Hold*
0	1	1	0	1	*Reset*
1	0	1	1	0	*Set*
1	1	1	X	X	*Illegal*
X	X	0	Last Q	Last $\overline{Q}$	*Hold*

Fig 10.68
Truth table for a Clocked RS latch.



GROUP TASK Activity
Draw up a complete 8 row truth table for the Clocked RS Latch in *Fig 10.67*. Complete this table to ensure the correct operation of the circuit. Make sure you understand this circuit before proceeding.

Illegal inputs of 11

Let us examine the first problem of illegal inputs of 11. If we examine the truth table for the clocked RS latch, we see that S and R values that alter the value stored in the latch are the inverse of each other. We can insert a not gate between the S input and the R input and then dispense with the R input altogether. This is only possible because of the inclusion of the clock circuitry. The hold conditions now exist only when the clock is 0. This revised latch design is known as a data latch, D-type latch or merely a D latch. The D or data input is now the same as the value stored within the latch and output at Q. Often this circuit is called a D-type flip-flop, however this is not strictly correct, as it has no edge trigger component.

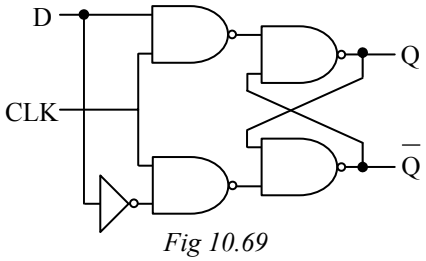


Fig 10.69
D latch circuit. The input at D is stored and output at Q.

Edge trigger component

An edge trigger circuit is used to ensure the flip-flop only changes state once for each tick (1) of the clock input. The clock changes state from a 0 to 1 and back to a 0, the aim is to perform the change of state within the flip-flop as the clock changes state. Each change of state is called an edge, in other words the flip-flop is triggered by either an up or positive edge or a down or negative edge.

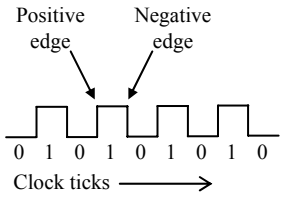


Fig 10.70
An edge is the point where the signal changes.

Edge trigger components are used on all types of flip-flops. For this discussion, we will consider the addition of edge trigger circuits to our D latch developed above. It is possible to design edge trigger circuits that respond to either the positive or the negative edge of the clock signal. For our discussion, we will examine a negative edge triggered D-type flip-flop.

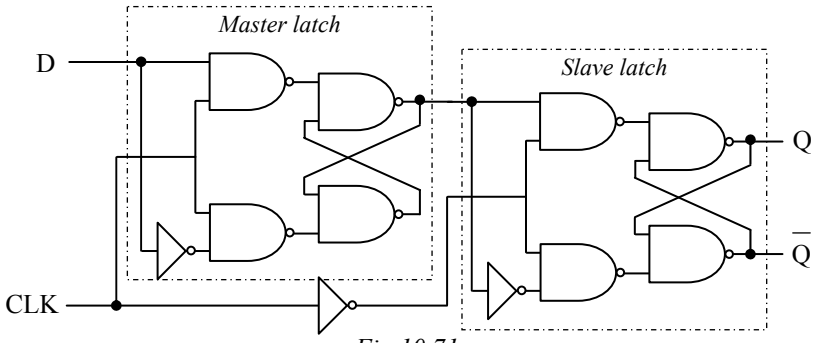


Fig 10.71
D-type flip-flop with negative edge trigger or more simply a D flip-flop.

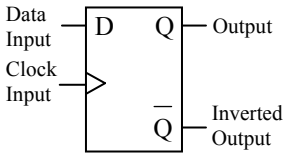


Fig 10.72
Symbol for a D flip-flop.

Edge triggered flip-flops consist of two latch circuits, a master and a slave. (Sometimes these flip-flops are called master-slave flip-flops). The inverter connecting the clock inputs is the key to the edge trigger function. If the clock input is 0 then data inputs at D will have no effect on the master latch. Consequently, there will be no change in the input into the slave latch, hence the stored value and output remains unchanged. This is precisely what is required when the clock sends a 0.

What about when a 1 is sent from the clock? In this case, the data input D into the master latch becomes active, as a result the value stored in the master latch can be altered to reflect the input. Whilst this is happening an inverted clock value of 0 has entered the slave latch, hence the slave latch is disabled and does not accept the input from the master latch. When the clock falls again to a 0 (the negative edge), the master latch is again disabled, however the inverted clock input is now 1 so the slave latch is active. Therefore, the output from the master latch can enter the slave latch and alter its contents. The result being that both latches are now storing the new data value D. As with the previous latch designs this value is output at Q.



GROUP TASK Investigation
Edge triggering is not a simple concept to grasp. There are various software simulations available both on the web and as part of software packages. Use one of these packages to investigate this concept.

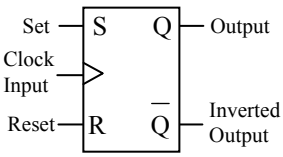


GROUP TASK Discussion
The explanation above is relevant to negative edge triggered flip-flops. Positive edge triggering can be accomplished by adding another inverter at the initial clock input. Explain how this system would operate.

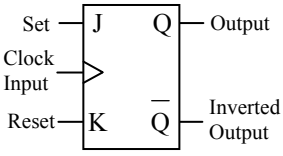
Other flip-flop designs

The D-type flip-flop is only one type of flip-flop. The RS flip-flop is a similar design however it does not combine the reset and set inputs into one, hence the illegal 11 input is possible. The JK flip-flop is a modification of the RS Flip-flop that uses a different system to overcome the illegal 11 problem. Essentially, the JK flip-flop uses feedback from the outputs to disable the input not being used.

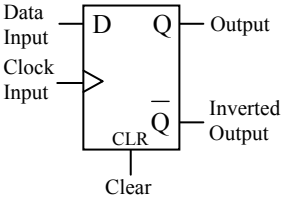
Other modified flip-flops have an input that causes the value stored to be returned to 0. This is useful when a large number of flip-flops need to be cleared simultaneously. In the next section, we look at shift registers. The clear function is particularly useful when flip-flops are to be used as part of registers within the CPU.



RS Flip-flop



JK Flip-flop



D Flip-flop with clear

Fig 10.73
Symbols for popular flip-flops



GROUP TASK Investigation
Collect catalogues from various electronics stores or access them on the web. Make a list of all the different types of flip-flop ICs available at these stores. If possible, obtain the pin assignments and draw a diagram similar to the NAND IC diagram in *Fig 10.41*.

Shift registers

Shift registers are used to store binary data. Registers are normally designed to store data in multiples of 4 bits. All shift registers allow for data to be moved into or out of the register. Designs are available to shift data left and/or right; these functions are necessary to perform binary multiplication and division. A shift of one bit to the right divides the contents by two; a shift of one bit to the left multiplies by two.

Essentially shift registers are groups of flip-flops connected together in a chain. The whole group is controlled by a common clock. This means that data can be shifted simultaneously throughout the register. If a clear function were present on the component flip-flops, this would also be controlled by a common circuit.

There are a number of ways of connecting flip-flops to provide for particular requirements. Some common arrangements include:

- Serial in – serial out
- Serial in – parallel out
- Parallel in – serial out
- Parallel in – parallel out
- Bi-directional
- Counters

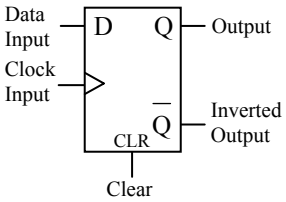


Fig 10.74
D Flip-flop with clear

For our purposes, we will examine the Serial in – serial out, Parallel in – parallel out and the Bi-directional shift register. In each case, we will use D flip-flops with clear inputs, however similar designs could be created using the other common flip-flop designs. For our discussion we will create four bit register designs but these designs could easily be extended to any number of bits.

Serial in – serial out shift register

This register is populated one bit at a time. To load 4 bits requires 4 clock ticks. As a bit is input from the left all other bits are shifted to the right, with the right hand bit being lost out of the register.

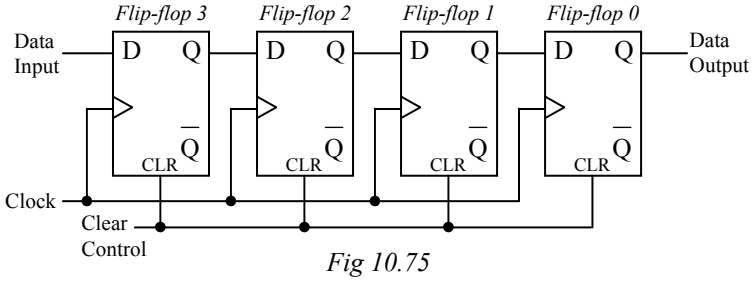


Fig 10.75
Serial in – serial out shift register

To load the register with data requires activating the clear control line. This is done by sending a 1; this sets all flip-flops to 0. The input data is then supplied to the data input line of flip-flop 3 one bit at a time. 1 bit enters the register for each tick of the clock. A bit destined for the right hand flip-flop (flip-flop 0) must travel through all preceding flip-flops to reach its destination. *Fig 10.76* is a desk check of this register receiving the bit pattern 1011.

Clock	Data Input	Flip-flop			
		3	2	1	0
0	1	0	0	0	0
1		1	0	0	0
0	1				
1		1	1	0	0
0	0				
1		0	1	1	0
0	1				
1		1	0	1	1

Fig 10.76
Desk check of the serial in – serial out shift register using the data 1011.



GROUP TASK Activity
Perform a desk check of the Serial in – serial out shift register. Assume the register contains the data 1011 and that it needs to be shifted out of the register and replaced by 0110.

Parallel in – parallel out shift register

This register receives and outputs data into and out of all its flip-flops simultaneously. The same number of input data lines are required as the number of flip-flops within the register, similarly for the data outputs. This is the normal structure for a storage register that cannot perform any processing.

Once the clock sends a 1, data is stored in all the flip-flops at the same time. In other words, one tick is required to fill the register with new data. The outputs from each flip-flop are available at all times. It is not necessary to shift data out via other flip-flops, as was the case with the serial out design.

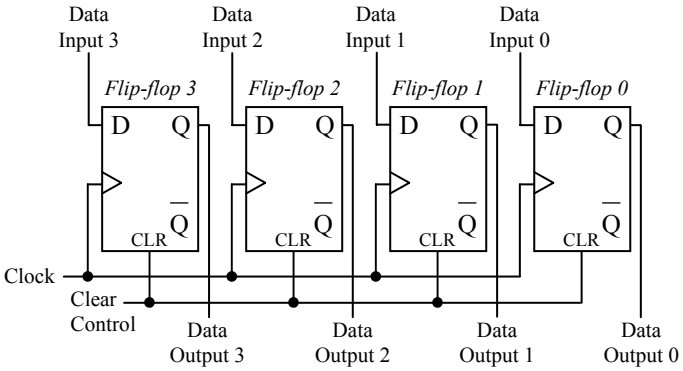


Fig 10.77
Parallel in – parallel out shift register

This register design is unable to alter the nature of its data as there is no link between the data lines of each flip-flop. This makes the design suitable as a storage register rather than a processing register. Because data can be clocked in and out of this register during a single clock tick, it is particularly suitable where fast access to data is required.



GROUP TASK Discussion
Describe the difference between serial and parallel transfer of data. Why do you think parallel transfer of data is the preferred method within the CPU? Explain.



GROUP TASK Activity
Study the two shift registers discussed above. Design a four bit shift register that could be described as a Serial in – parallel out shift register.



GROUP TASK Discussion
Multiplexers are used to combine signals from multiple sources and send them down a single line and vice versa. Do any of the following shift registers perform this function? Explain your answers.

- Serial in – serial out
- Serial in – parallel out
- Parallel in – serial out
- Parallel in – parallel out

Bi-directional shift register

The above serial in – serial out register can only shift data to the right, each shift effectively dividing the number by two. The Bi-directional shift register is able to shift data in either direction to allow both multiplication and division to take place.

This register contains an extra control line (left/right control) used to indicate the direction of shift. If the control line is sent a 1 then the data is shifted one position to the right, with the data input being placed into the left hand flip-flop. If the left/right control is sent a 0 then the data is shift one position to the left, with the data input being placed into the right hand flip-flop.

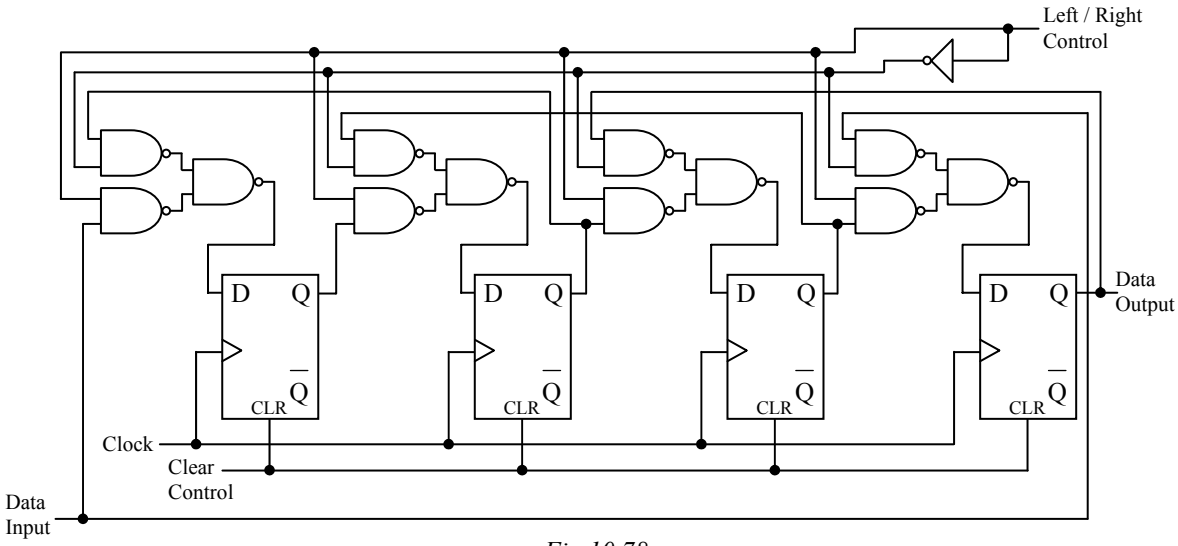


Fig 10.78
Bi-directional shift register.

The logic of this flip-flop is quite involved and requires intensive examination to understand completely. Let us examine a left hand shift using this circuit. Firstly, the left/right control is sent a 0, indicating a left shift. This input is reversed by the inverter on the second left/right control line. As a result, each set of 3 NAND gates, controlling the inputs into each flip-flop, receives a 1 at the top NAND gate and a 0 at their bottom NAND gate. Consider the bottom NAND gate preceding each flip-flop, as it has received a 0 then its output must be a 1 regardless of the value in its other input, effectively these gates have been disabled. The upper NAND gates, which received a 1 from the left/right control, are enabled. A 1 from their other input will result in a 0 output and a 0 results in a 1 output. Following the source of the other input to the upper NAND shows it comes from the output of the flip-flop to the right, which is correct for a left hand shift. The third NAND gate preceding each flip-flop input effectively reverses the input back to its correct state for storage in the flip-flops. Of course, all this can only occur if the clock has sent a 1 to each flip-flop. Notice that the right hand flip-flop will gain its new data from the data input line.



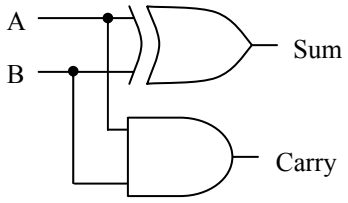
GROUP TASK Discussion
The above paragraph describes a left shift of the above register. A right shift occurs when the left-right control line sends a 1. Describe the processing involved in a right shift.



HSC style questions:

Question 1.

Consider the following binary half adder circuit:



(a) Identify the two logic gates used in the above half adder circuit.

(b) Construct a truth table for the above half adder circuit.

(c) Create a circuit for a full adder.

(d) Explain how a circuit to add two 8-bit binary numbers could be constructed.

Question 2.

Describe the essential feature of flip-flops that allows them to store binary digits.

Suggested solutions

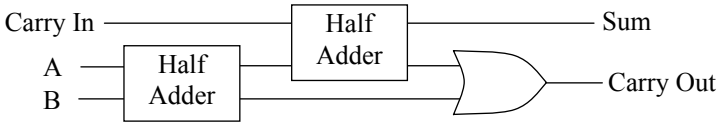
Question 1.

(a)  is an AND gate and  is the symbol for an XOR gate.

(b)

A	B	Carry	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

(c)



(d) When we add any 2 single bits from a number which has multiple bits, there can be a carry which needs to be carried forward to be included in the addition of the next 2 bits in the number. In effect three bits are input and added resulting in 2 bits being output.

This circuit requires that the carry from the previous addition of the lower order bits in the two 8-bit numbers is carried forward into the addition of the next 2 equivalent bits. As there are 8 pairs of bits then 8 full adders must be connected together – the carry out from each full adder connecting to the carry in of the full adder for the next higher significant bits.

Question 2.

The latch component of the flip-flop. This component contains two gates with the output of each linked back to one of the inputs of the other. This provides feedback to the gates so that the latch is able to remain in a stable state. There are two possible states representing the binary digits 0 and 1.

SET 10E

1. A flip-flop:
   (A) is used to store a binary digit.
   (B) can store a single character.
   (C) is used to control the operation of a latch.
   (D) is used within permanent memory.

2. A latch:
   (A) uses flip-flops as part of their design.
   (B) always has a clocking component.
   (C) uses feedback from the outputs of two logic gates.
   (D) is used to ensure data is not overwritten.

3. Edge triggering is used to ensure:
   (A) flip-flops only change state when the clock is sending a 1.
   (B) the value stored in the flip-flop cannot be altered.
   (C) the flip-flop only changes state once for each tick of the clock.
   (D) illegal inputs are not processed.

4. An eight bit serial in – parallel out shift register would require:
   (A) 8 clock ticks to load the register and 8 to read the register.
   (B) 1 clock tick to load the register and 1 to read the register.
   (C) 8 clock ticks to load the register and 1 to read the register.
   (D) 1 clock tick to load the register and 8 to read the register.

5. A flip-flop always contains the following components:
   (A) latch, clocking and edge triggering components.
   (B) NAND and NOT gates.
   (C) a single bistable element.
   (D) All of the above.

6. A shift register can be described as:
   (A) a group of flip-flops connected together.
   (B) circuits that store individual bits.
   (C) circuits that can multiply and divide.
   (D) the basic building block for a flip-flop.

7. The clocking component of a flip-flop:
   (A) allows multiple access to the flip-flop.
   (B) allows synchronisation of multiple flip-flops.
   (C) removes the possibility of illegal inputs.
   (D) can be negative or positive edge triggered.

8. A four bit parallel in – serial out shift register would require:
   (A) 1 data input and 4 data outputs.
   (B) 4 data inputs and 1 data output.
   (C) 1 data input and 1 data output.
   (D) 4 data inputs and 4 data outputs.

9. RS Flip-flops perform unpredictably if they receive inputs of 11. Why is this the case?
   (A) Inputs R and S must be the inverse of each other at all times.
   (B) One of the two feedback lines will conflict when the illegal inputs arrive.
   (C) Circuits are not perfect, this is a problem that cannot be overcome.
   (D) This is a consequence of adding a clocking component.

10. Which type of shift register is unable to perform either left or right shifts?
    (A) parallel in – serial out
    (B) serial in – serial out
    (C) parallel in – parallel out
    (D) bi-directional

11. Describe how feedback is used to create circuits that are able to store binary digits. Draw a diagram of such a circuit to assist with your description.

12. Flip-flops require clocking components to operate successfully. Draw a diagram of an RS-NOR latch that includes a clocking component. Describe the problem solved by adding a clocking component. How does the clocking component solve this problem?

13. Flip-flops require edge triggering components. What is the purpose of edge triggering circuitry? What is the difference between positive and negative edge triggering?

14. Shift registers are used to store binary data, however they can perform other useful processes as well. Describe 2 other processes that shift registers can accomplish. How are these processes accomplished?

15. Each flip-flop contains a clocking component. When flip-flops are connected to form shift registers the clocking components perform a vital function. Describe this function.

PROGRAMMING OF HARDWARE DEVICES

Computer systems require input from outside sources. This input may be acquired from traditional input devices such as the keyboard, mouse or CD-ROM or it may come from external sensors that can react to changes in the environment such as light, temperature, flow, pressure or movement. Similarly, output from the computer system requires conversion into forms that are understandable by the output device. Most computer systems include monitors, printers and sound output. Other systems include actuators that cause some mechanical action e.g. moving a robotic arm or unlocking a door.



Sensor
Any device that converts input from the environment into a digital signal. This signal is interpreted by the software.



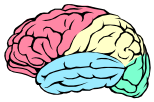
Actuator
An output device that performs a mechanical action. Actuators receive digital signals from software.

Software developers must be able to send and receive data to and from these input and output hardware devices. In this section, we examine the data streams that are required for the control of hardware devices by software.

Input and output data streams from and to sensor and other devices

The data stream received from input devices and sent to output devices include two types of information; the actual data together with information to ensure its correct delivery. The format commences with header information followed by the data itself and concluding with trailer information. Both the header and trailer information are used to ensure the data is delivered correctly.

Manufacturers of hardware devices produce specifications detailing the precise nature and format of the data streams sent to and from their products. Similarly, hardware specifications detail the format of the data required to control these devices. It is the responsibility of the software developer to interpret and create these data streams correctly.



Consider an older PS2 ball mouse:

There are two different types of sensor built into a mouse; sensors to detect motion and sensors to detect the state of the buttons. A typical mouse with two or three buttons would have a separate sensor for each button together with sensors for movement. Today the movement sensors are essentially small yet complex digital cameras which detect changes in the image to sense motion. Older designs used mechanical rotating sensors in conjunction with a rubber ball to detect movement in both X and Y directions (see *Fig 10.79*). We will consider one of these older designs which connect to the computer using a PS2 serial port (rather than USB) as the data stream sent from the mouse is much simpler to understand.

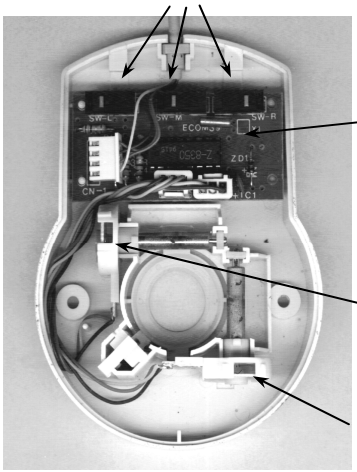


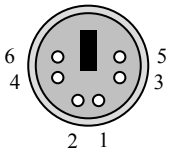
Fig 10.79
Underside of mouse showing input sensors.

Mouse buttons are simple switches; either they are up or down. The sensors for X and Y direction detect and send a series of 1s and 0s to the mouse's IC. The IC converts the signals from the switches and X and Y direction sensors into coded form, which is sent to the computer via a PS2 serial port.



GROUP TASK Practical Activity
Remove the cover from an old mouse (not necessarily a ball mouse). Identify each of the sensors within the mouse. Can you identify any of the other components within the mouse?

On old IBM compatible computers, a PS2 port was used to receive data from the mouse. The PS2 port was developed to free up valuable serial ports prior to the widespread acceptance of USB. PS2 ports receive synchronous data. This means the rate of data transmission occurs in time with the clock signal, one bit is sent from the mouse for each tick of the clock. *Fig 10.80* shows the pin assignments for PS2 connectors; pin 1 is used for data and pin 5 for the clock signal.



Pin no.	Name
1	Data
2	Reserved
3	Ground
4	Power
5	Clock
6	Reserved

Fig 10.80
PS2 female connector and pin assignments.

Let us now examine the format and nature of the data stream sent from the mouse. When a mouse sends information, it sends three consecutive data packets. These data packets contain information about the state of each button and the movement that has occurred in both X and Y direction since the last transmission. Different brands of mouse may use different formats or include additional data packets to allow the inclusion of added features. For our discussion, we will examine the input data stream from a standard two button PS2 mouse.

Packet No.	Bit number							
	7	6	5	4	3	2	1	0
1	Yv	Xv	Yd	Xd	1	0	Bl	Br
2	X7	X6	X5	X4	X3	X2	X1	Y0
3	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0

Yv Overflow in Y direction.
Xv Overflow in X direction.
Yd Direction of movement in Y direction (1 means left, 0 means right).
Xd Direction of movement in X direction (1 means up, 0 means down).
Bl Left button state (1 means down, 0 up).
Br Right button state (1 means down, 0 up).
X0-X7 Number of bits moved in X direction since last transmission.
Y0-Y7 Number of bits moved in Y direction since last transmission.

Fig 10.81
Data format for a two button PS2 mouse.

Each packet of data is eleven bits in length containing a start bit, eight data bits, an odd parity bit and finally a stop bit. As there are three data packets, a total of 33 bits are required for each transmission. The format of the data is shown in *Fig 10.81*. Movement in each direction is sent using 8 bits. Values in the range 0 to 255 are possible. If the mouse has been moved exceptionally fast then an overflow can occur in which case the appropriate overflow bit is set to 1.

Fig 10.82 shows the input stream from a typical transmission. For this mouse the start bits are always zeros and the stop bits are always ones.

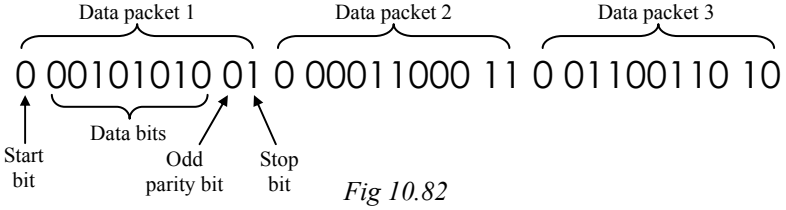


Fig 10.82
Data stream for a typical PS2 mouse transmission.



GROUP TASK Discussion
Examine the example input data stream in *Fig 10.82*. What can you say about the state of the mouse buttons? Describe the movement of the mouse?

All data packets are made up of three distinct components; header information, data and trailer information. The data will often contain control characters used to cause some predefined action by the hardware e.g. moving a robotic arm or moving a printer head to the left hand side. Let us consider each of these three components in detail.

Header information

The information preceding the data is known as the header. For communication with hardware devices within the computer system or connected directly to one of its ports, the header is often just a single start bit. This bit signals to the receiving device that a new data packet is commencing.

Data characters

The actual data being sent. This is the stream of data that is used by the receiving device. Often control characters are included within this data stream. For example, a modem requires control sequences to activate its internal functions, such as instructions to go off-hook or dial a number.

Trailer information

The information following the data is known as the trailer. Trailer information includes error checking bits and stop bits. There are various error checking techniques that can be used, often a simple parity bit is used. A stop bit is often used to indicate the end of the data packet.

Processing of the data stream

Once a data stream has been received from a hardware device it will require processing by the computer. The software used to perform this function must be able to understand the precise format of the data stream. Control characters will need to be isolated and acted upon and strings of data will need to be interpreted into forms that can be used by the system. The processing required is dependant on the particular hardware device.



Consider the following:

PABX systems are used to control the telephone systems in large organizations with multiple lines and extensions. In large hotels, the PABX is used for many administrative functions. For example, housekeeping staff are able to indicate when a room has been cleaned using the telephone system, guests can activate do not disturb status, etc. The hotel's computerised property management system (PMS) interfaces with the PABX.



Fig 10.83
PABXs are used to control the operation of telephone systems.

Different PABX systems use different formats for messages. Software developers need to create interfaces that are able to accept messages from all the commonly available models of PABX. Let us examine the format of a typical data stream between the PMS and a fictitious model PABX we'll call the PEC2010. The PEC2010 is connected to the PMS computer via a serial port.

The PEC2010 uses the same general format for both messages it sends to the PMS and messages it receives from the PMS. Each message is comprised entirely of ASCII characters. The header commences with the STX or start of text character, which is followed by two characters indicating the model of the PABX, in this case, P2, and a single digit indicating the version; for hotel systems the version is always a 5. Hence the header for all messages within the hotel will be STX P25 which translates to the ASCII values 2 80 50 53.

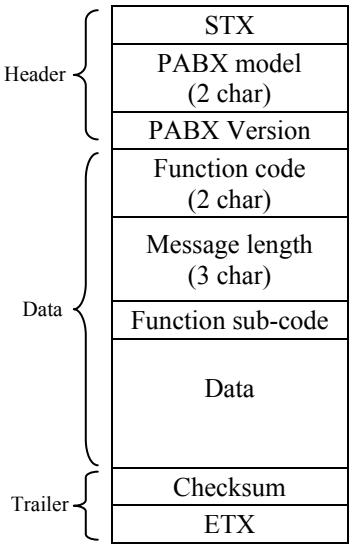


Fig 10.84
General message format for the PEC2010 PABX.

Following the header information the data to be processed commences. This data commences with a two character function code, followed by three digits indicating the length of the message in characters, followed by a function sub-code; a total of six characters. The message length is the length of the message including the header but excluding the initial STX character and final ETX character.

The trailer information for each message contains a 1 byte checksum followed by the ETX or end of text character which has an ASCII value of 3. After each message has been sent, the receiver sends an ACK (acknowledge) if the message was received correctly or a NAK (negative acknowledge) if an error was detected. If a NAK is received then the message is resent.

Let us now examine the format for a particular function: Guests need to be charged for calls they make according to the length of the call and the destination of the call, these messages are transmitted from the PABX to the PMS for inclusion on the guest's account. This function is called the 'Call Detail' function and is assigned the function code 24. Within this function there are a number of sub-codes to deal with internal calls, operated assisted calls and normal direct calls. The function sub-code we require is coded as a 4. *Fig 10.85* describes the message format of "Call Detail" function 24 sub-code 4. Each character is coded and sent using ASCII including the value of the checksum.

1	STX
2	P
3	2
4	5
5	2
6	4
7	0
8	5
9	4
10	4
11 12 13 14	Extension number
15 16 17	Line number
18 … 37	Called number (Space padded)
38 39 40 41 42 43	Call start time (Hours, min, sec)
44 45 46 47 48 49	Call duration (seconds)
50 51 52 53 54	Metered pulses
55	Checksum
56	ETX

Fig 10.85
Call detail message format for the PEC2010 PABX.

Let us examine an example message coded using the specification described in *Fig 10.85*. Mr Bloggs in room 305 makes a call to 0291234567 at 17 minutes and 23 seconds after 6pm, the call lasts 256 seconds. In this particular hotel there are no metered pulses and the extension numbers are the same as the room numbers. The message sent would be as follows: [STX] P2 5 24 054 4 0305 006 0291234567 [10 spaces] 181723 000256 00000 [Checksum] [ETX]. When converted to ASCII codes, we get the data stream: 2 80 50 53 50 52 48 53 52 52 48 51 48 53 48 48 54 48 50 57 49 50 51 52 53 54 55 32 32 32 32 32 32 32 32 32 32 49 56 49 55 50 51 48 48 48 50 53 54 48 48 48 48 48 [Checksum] 3.

The PMS software must be able to interpret this message. To do this requires a procedure or function that firstly is able to ensure the message was received correctly and secondly is able to extract each field from within the data stream. Imagine the data stream is stored as a string in the variable InStream, *Fig 10.86* describes a possible algorithm for checking the input data stream has been received correctly.

```
BEGIN CheckInstream(InStream,OKFlag)
  Set OKFlag to False
  Set LenStream to length of InStream – 2          ┐ Check message
  Set MesLength to value of chars 7 to 9 in InStream │ length is correct
  IF LenStream = MesLength THEN                    ┘
    Set Checksum to 0                              ┐
    Set CharCount to 2                             │
    WHILE CharCount <= MesLength                   │ Ensure Checksum
      Add ASCII value of character in position Charcount to Checksum │ is correct
      Increment CharCount                          │
    ENDWHILE                                       │
    Set CheckStream to ASCII value of second last char in InStream │
    Set Checksum to last two digits of hex value of Checksum │
    IF CheckStream = Checksum THEN                 ┘
      Set OKFlag to True   <- If both length and checksum are correct
    ENDIF
  ENDIF
END CheckInstream
```

Fig 10.86
Algorithm to check the length and Checksum of an input message.



GROUP TASK Discussion
Examine the algorithm in *Fig 10.86*. Describe how the Checksum is calculated. Why do you think the last two hexadecimal digits are used rather than the first two digits?

For this particular 'Call Detail' function the PMS software requires the extension number, called number, time the call was made and the length of the call. The PMS must also calculate and store the call charge. All this data will eventually be stored in the guest's account. An algorithm to perform this processing is reproduced below.

```
BEGIN ProcessCallDetail(Instream,CallRecord)
  CallRecord.Room = Value of chars 11 to 14 of Instream
  CallRecord.Phone = Value of chars 18 to 37 of Instream
  CallRecord.Start.Hour = Chars 38 to 39 of Instream
  CallRecord.Start.Min = Chars 40 to 41 of Instream
  CallRecord.Duration = Value of chars 44 to 49 of Instream
  CallRecord.Charge = CallCharge(CallRecord.Phone,CallRecord.Duration)
END ProcessCallDetail
```

Fig 10.87
Algorithm to extract the call details from the input data stream.



GROUP TASK Discussion
Examine the algorithm in *Fig 10.87*. What type of data structure is the CallRecord variable; describe this structure? What do you think the CallCharge function does? Could the ProcessCallDetail procedure have been written as a function?

Control systems

A control system is made up of sensors that obtain data from the environment and actuators that are able to perform output functions under the direction of a controller. In this course, we are concerned with control systems where the controller is a computer system. The sensors provide the input into the system, the controller performs the processing and as a consequence directs the actuators to perform some physical task. There are two broad categories of control system; open control systems and closed control systems.



Fig 10.88
Hotel door locks are examples of open control systems.

An open control system is not able to react to its environment. For example, in many hotel rooms the lock on each room is activated using a magnetic stripe card. The card is encoded with details of the guest's arrival and departure time together with a unique code particular to the room. A sensor in the lock reads the magnetic stripe. The details are processed by a controller within the lock. If they are correct, an actuator unlocks the door. There is no feedback to the controller, and in this case, none is required.

A closed loop system is able to react to its environment. In other words, the sensors provide feedback so that the controller can cause the actuators to react to changes in the environment. For example, the antilock braking system on a car senses when a wheel has locked (stopped turning) and causes the brake on that wheel to be momentarily disengaged. The actuators, in this case, are a series of valves that release hydraulic pressure in the system. A closed loop exists between the sensors and the actuators.



Fig 10.89
Antilock brake systems are closed control systems.

Production lines that use robots to perform repetitive tasks utilise both open and closed control systems depending on the application. Sensors that are able to detect movement, light and motion provide the input into these systems. Often the actuators are electric motors that cause movement of portions of the robot. Robots that are part of a closed system are often called intelligent robots as they able to react to their environment. Robots that are part of an open system are known as dumb robots as they will continue to carry out their task regardless of changes in their environment.



GROUP TASK Discussion
Many household appliances are control systems. Discuss the operation of a number of household appliances. What sensors and actuators are present in these appliances? Classify each system as either an open or closed loop system.



HSC style questions:

Question 1.

On many major Sydney roads there are electronic messages. The EDI model 2411 is a particular model of scrolling electronic marquee that is 200mm high and is able to accommodate up to 24 characters.



You are currently in the process of writing a software application to communicate simultaneously with a set of EDI 2411 scrolling marquees and hence you need to understand the technical aspects of the data stream required by the unit.

Refer to the EDI 2411 technical information in *Fig 10.90* on the following page to assist when answering the following.

(a) Create a data stream to display the words "HELLO WORLD" on all the displays, the word "HELLO" should be flashing. The message is to appear from the top of the displays, pause for two seconds, and then scroll off the bottom of each display.

(b) A series of text messages are stored in an array called Messages. The last item in the array is a sentinel, namely "ZZZ". Write an algorithm that creates a data stream to display all of these messages, in the order they are stored, on all the EDI 2411 displays.

Suggested solutions

Question 1.

(a) <CTRL-D><CTRL-G> "000" <CTRL-J> "01" <CTRL-N> <CTRL-O> "HELLO" <CTRL-O> "WORLD" <CTRL-R>

(b)
```
BEGIN Displaymessages
    Index=1
    Stream=""
    Message=Messages(Index)
    WHILE Message ≠ "ZZZ"
        Stream = Stream & <CTRL–D><CTRL-G> "000" <CTRL-J>
        Stream   = Stream & Integer part of Index /10
        Stream = Stream & Remainder of Index /10
        Stream = Stream & Messages (Index)
        Stream = Stream & <CTRL-R>
        Index = Index + 1
        Message = Messages(Index)
    ENDWHILE
    Send stream to displays
END
```



GROUP TASK Activity

There are 10 EDI 2411s along a freeway with addresses 001 to 010. Write an algorithm which cause each marquee to cycle through an array of 10 messages such that every marquee is displaying a different message.

Message Protocol:

<start-of-msg> <addr> <msg no> <text> <end-of-msg>

where

ITEM	VALUE (DECIMAL)	1 FUNCTION
<start-of-msg>	4	Standard ASCII character 'EOT', also known as a 'control D', this character must be the first character of each message.
<addr>	7 48-50 48-57 48-57	Standard ASCII character 'BEL', also known as a 'control G', followed by three standard ASCII decimal digits representing the address of the display as set on the display from '001' to '255'. The 'BEL' character must always be the second character of each message. The address must always be a three digit number (use leading zero's if needed). If address '000' is entered, all marquees will display the message.
<msg no>	10 48-51 48-57	Standard ASCII character 'LF', also known as a 'control J', followed by two standard ASCII decimal digits representing the message number from '01' to '32'. The 'LF' must always appear immediately after the address and the message number must always be a two digit number (use a leading zero if needed).
<text>	01-126	Alpha/Numeric data to be displayed in ASCII 'printable' characters and ASCII control characters to specify display options (see below).
<end-of-msg>	18	Standard ASCII character for 'DC2', also known as a control R, this character must be the last character of each message.

'BLOCK' Commands for Moving Marquees:

These commands should be placed at the beginning of a block of text. All of these commands consist of a single ASCII 'Control' character. They are used to control the visual effect of the entire block of text. If no block command is used the text will scroll onto the display from left to right, pause for two seconds then scroll off to the right.

'BLOCK' COMMAND	VALUE (DECIMAL)	2 FUNCTION
'Control I' (<**TAB**>)	9	This command causes the text that follows to 'JUMP' onto the display where the entire message appears at once. The message will then pause for two seconds and then disappear at once.
'Control K' (<**VT**>)	11	This command causes the text that follows to scroll rapidly 'UP' onto the display from the bottom. The message will then pause for two seconds and then scroll rapidly 'UP' off of the display.
'Control N' (<**SO**>)	15	This command causes the text that follows to scroll rapidly 'DOWN' onto the display from the top. The message will then pause for two seconds and then scroll rapidly 'DOWN' off of the display.
'Control]' (<**GS**>)	29	This 'BLINK' command causes the text that follows to appear starting with the middle horizontal row and build rapidly onto the display towards both the top and bottom. The message will then pause for two seconds and then disappear rapidly in reverse order.

'TOGGLE' Commands for Moving Marquees:

These commands should be placed at the beginning and at the end of a block or portion of a block of text. All of these commands consist of a single ASCII 'Control' character. They are used to control the visual effect of the block of text that is between them.

'TOGGLE' COMMAND	VALUE (DECIMAL)	3 FUNCTION
'Control O' (<**SI**>)	15	This pair of commands causes the text between them to 'FLASH' on and off at approximately ½ second on and ½ second off.
'Control \' (<**FS**>)	28	This pair of commands causes the text between them to appear in double 'WIDE' format.

Fig 10.90
Abridged extract from the technical manual for the EDI 2411 scrolling marquee.

Question 2.

A model helicopter is controlled via RF signals from a remote control transmitter/receiver. Within the remote control all messages are represented and processed as ASCII characters.

The helicopter has 3 motors, one controls its forward speed, one causes it to rotate either left or right and another causes it to either climb or dive. A series of commands can be combined into a single data stream. Each data stream commences with an STX or start of transmission character (ASCII code 2), followed by the total number of characters in the command sequence, and then the actual command sequence. After the commands an ETX or end of transmission character (ASCII code 3) terminates the data stream.

The motors respond to the following commands sent from the remote control.

Command	Description
R0	Rotate left
R1	Rotate right
R	Stop rotation
A0	Altitude up, meaning helicopter will climb.
A1	Altitude down, meaning helicopter will dive.
A	Maintain altitude.
S*n*	Speed where n is an integer from 0 (stop motor) to 9 (full speed)

For example the data stream <STX>6R0A1S3<ETX> causes the helicopter to rotate left and dive at a speed of 3.

The helicopter transmits a data stream specifying its position every second using data collected from its sensors. There are always 9 characters of data. The first 3 characters represent the current bearing from 000 to 359 degrees. The second 3 represent the altitude from 000 to 999 metres. And the final 3 characters represent the distance travelled in the previous second from 000 to 999 metres. This data is preceded by an STX character and is terminated with an ETX character. Hence each transmission received by the remote control is 11 characters long. For example the data stream <STX>180010020<ETX> means the helicopter is flying due south at a height of 10 metres and travelled 20 metres in the last second.

Wind conditions alter continuously and have a significant effect on the helicopter's bearing, speed and altitude. As a consequence the three motors need to be constantly altered in order for the helicopter to maintain steady flight.



GROUP TASK Activity
Assuming there is no wind or other air currents describe a likely sequence of data streams which will cause the helicopter to rise to an altitude of 20 metres and then return smoothly to ground level and stop.



GROUP TASK Discussion
Discuss how the three motors can be used to keep the helicopter flying on a particular course and at a particular altitude in windy conditions.

An algorithm is being developed to maintain the helicopter on a constant bearing, and at a constant altitude and speed. The main program has been written and is reproduced below.

```
BEGIN ControlChopper
    Get ConstBearing in degrees, ConstAltitude in metres, ConstSpeed in metres/sec
    LastSpeed = 5
    WHILE Chopper is in the air
        REPEAT
            Wait
        UNTIL stream arrives from chopper
        Receive data stream into InStream
        ConvertInStream(InStream, Bearing, Altitude, Distance)
        OutStream = RotationCommand(ConstBearing, Bearing)
        OutStream = Outstream & AltitudeCommand(ConstAltitude, Altitude)
        OutStream = Outstream & SpeedCommand(ConstSpeed, Distance, LastSpeed)
        Outstream = <STX> & length of Outstream & Outstream & <ETX>
        Transmit OutStream to chopper
    ENDWHILE
END ControlChopper
```

(a) Create an algorithm for the ConvertInStream routine.

(b) Assume the helicopter is to be maintained on a constant bearing of 90 degrees, an altitude of 30 metres and a speed of 20 metres per second.

Write down a suitable data stream to be transmitted to the helicopter after each of the following data stream sequences. Justify your responses.

I. Transmit to helicopter: <STX>6R0A0S5<ETX>
Receive from helicopter: <STX>100035020<ETX>

II. Transmit to helicopter: <STX>6R1A0S2<ETX>
Receive from helicopter: <STX>265030015<ETX>

(c) Create an algorithm for each of the following routines.

(i) SpeedCommand

(ii) RotationCommand

(iii) AltitudeCommand

Suggested solutions

Question 2.

(a)
```
BEGIN ConvertInStream(InStream, Bearing, Altitude, Distance)
        Bearing = value of chars in position 2-4 of InStream
        Altitude = value of chars in position 5-7 of InStream
        Distance = value of chars in position 8-10 of InStream
    END ConvertInStream
```

(b) I. <STX>2A1<ETX>. The helicopter is already turning left so no need for a rotation command. The helicopter is too high so need to dive (A1). The current speed is perfect so no need to alter the engine speed of the speed motor. There are two characters in the data stream, so commence with 2.

II. <STX>5R0AS3<ETX>. As 265 degrees is closer anti-clockwise to 90, we rotate left (R0). The altitude is perfect so maintain altitude (A). The chopper is flying slow, so increase engine speed by 1 from S2 to S3. There are 5 characters within the stream so commence with 5.

(c) (i)
```
BEGIN SpeedCommand(ConstSpeed, Distance, LastSpeed)
    IF ConstSpeed = Distance THEN
        SpeedCommand = ""
    ELSEIF ConstSpeed > Distance THEN
        IF LastSpeed<9 THEN
            Add 1 to LastSpeed
            SpeedCommand = "S" & LastSpeed
        ENDIF
    ELSE
        IF LastSpeed >0 THEN
            Subtract 1 from LastSpeed
            SpeedCommand = "S" & LastSpeed
        ENDIF
    ENDIF
END SpeedCommand
```

(ii)
```
BEGIN RotationCommand(ConstBearing, Bearing)
    IF ConstBearing = Bearing THEN
        RotationCommand = "R"
    ELSE
        Temp = ConstBearing – Bearing
        IF Temp < -180 THEN
            Add 360 to Temp
        ELSEIF Temp > 180 THEN
            Subtract 360 from Temp
        ENDIF
        IF Temp > 0 THEN
            RotationCommand = "R1"
        ELSE
            RotationCommand = "R0"
        ENDIF
END RotationCommand
```

(iii)
```
BEGIN AltitudeCommand(ConstAltitude, Altitude)
    IF ConstAltitude = Altitude THEN
        AltitudeCommand = "A"
    ELSEIF ConstAltitude > Altitude THEN
        AltitudeCommand = "A0"
    ELSE
        AltitudeCommand = "A1"
    ENDIF
END AltitudeCommand
```



GROUP TASK Discussion

Work through the operation of the ControlChopper algorithm including the subroutine calls and suggest suitable methods of testing the system.

SET 10F

1. A device that provides input into a control system is called:
   (A) an actuator.
   (B) a sensor.
   (C) a data stream.
   (D) a controller.

2. A device that causes some output from a control system is called:
   (A) an actuator.
   (B) a sensor.
   (C) a data stream.
   (D) a controller.

3. A stepping motor is able to turn precise amounts. A stepping motor is an example of:
   (A) an actuator.
   (B) a sensor.
   (C) a data stream.
   (D) a controller.

4. Both input and output data streams from and to hardware devices are made up of:
   (A) sensors and actuators.
   (B) ASCII characters.
   (C) header, data and trailer information.
   (D) control and data information.

Consider the following scenario when answering questions 5 and 6:

The automatic doors often used at the entrance to buildings and supermarkets are control systems. Stepping on a mat in front of the door causes the door to open. After a set period of time the doors close.

5. It is likely this system contains:
   (A) motion sensors and electric motors as actuators.
   (B) pressure actuators and electric motors as sensors.
   (C) a mechanical device that causes the door to open.
   (D) pressure sensors and electric motors as actuators.

6. Assume that the doors always close in a set period of time. This system is an example of:
   (A) an open control system.
   (B) a closed control system
   (C) a synchronous system.
   (D) an asynchronous system.

Consider the following when answering questions 7, 8 and 9:

A scanner sends a separate block of data for each pixel scanned. The block contains a code in the range 0 to 255 for red, green and blue in that order. Each data block is preceded by a start bit and followed by a 1 byte checksum and a stop bit.

7. The number of bits in the header, data and trailer respectively is:
   (A) 1, 3 and 2.
   (B) 8, 24 and 16
   (C) 1, 24, 2
   (D) 1, 24, 9

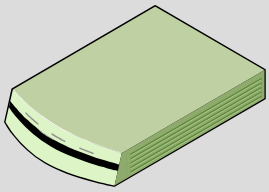
8. If data values of 0, 0, 0 is black and 255, 255, 255 is white then 0, 0, 255 would be which colour:
   (A) red
   (B) green
   (C) blue
   (D) yellow

9. How many unique colours can be represented using this system?
   (A) $3 \times 3 \times 3$
   (B) $255 \times 255 \times 255$
   (C) $8 \times 8 \times 8$
   (D) $256 \times 256 \times 256$

10. A robot on a production line is used to weld two components together. This robot continues to weld even though no further components have arrived. This control system is an example of:
    (A) an intelligent robot in an open loop control system.
    (B) an intelligent robot in a closed loop control system.
    (C) a dumb robot in a closed loop control system.
    (D) a dumb robot in an open loop control system.

Use the following information to answer questions 11 and 12:

Most modems use a set of commands known as the 'Hayes AT command set'; these commands are used to control the operation of the modem. Each command string is preceded by the AT command; AT means "attention modem, this command is for you - don't send it down the line". A carriage return signifies the end of the AT command. Data not preceded by AT is sent down the phone line. The following table describes a sub-set of the Hayes AT command set:



Command	Description
A	Answer mode. Go off hook and commence handshaking.
Dx	Go off hook and dial the number x. e.g. ATD91234567 dials the number 91234567.
H	Hang up.
Mx	M0 speaker always off, M1 speaker off once online (default), M2 speaker always on.
P	Use pulse dialling.
T	Use tone dialling.

Command	Description
Ex	E0 disables echo to local terminal. E1 enables echo to local terminal (Default).
Fx	Set speed of transmission. F0 auto detect, F13 is 14400bps, F19 is 28800 bps.
Lx	Speaker volume control. L1 low, L2 medium, L3 high.
Xx	X2 dialtone detect on and busy tone detect off, X3 dialtone detect off and busy tone detect on, X4 dialtone detect on and busy tone detect on.
Z	Reset modem to default values.

11. Write a data stream that would cause a modem to perform the following tasks:

    (a) Go off hook in answer mode at a speed of 28800bps.

    (b) Dial your home phone number using pulse dialling.

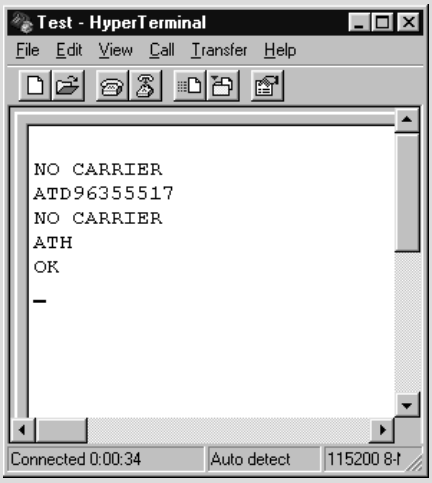
    (c) Dial your home phone number using tone dialling and send the word 'Hello'.

    (d) Ignore both dial and busy tones then dial your home phone with the speaker volume set to high.

12. Use a terminal emulation program, such as hyper terminal, to try a number of these commands with your own modem.

    What is the result of the ATE command?

    Does your terminal emulation program have menu options that can be used to accomplish some of the tasks performed by the AT command set? Explain.



Use the following information to answer questions 13, 14 and 15:

The Epson FX series of dot matrix printers is controlled by a series of control codes; the table below details some of the available codes. Some of the control codes are single ASCII characters; many use the ASCII escape character (decimal 27) to indicate the information is a command to the printer rather than text to be printed.

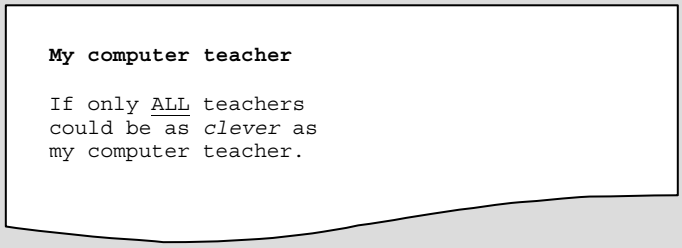
ASCII	Decimal	Description
BEL	7	Cause bell to sound
LF	10	Line feed (Move paper up 1 line)
CR	13	Move print head to left hand side
ESC – 0	27 45 48	Cancel underline
ESC 5	27 53	Cancel italic mode
ESC F	27 70	Cancel bold mode

ASCII	Decimal	Description
BS	8	Backspace. Move left one char.
FF	12	Form feed. Load a new page
ESC C n	27 67 n	Set page length in lines (n=1-127)
ESC – 1	27 45 49	Select underline
ESC 4	27 52	Select italic mode
ESC E	27 69	Select bold mode

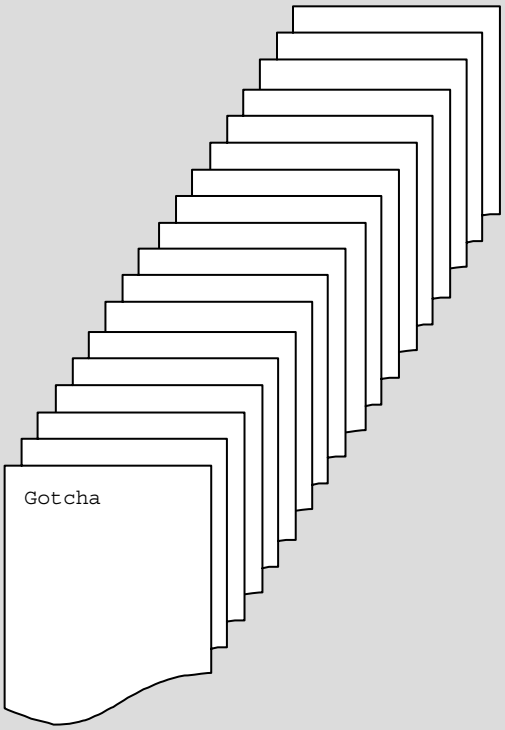
13. Explain the result of sending the following data stream to this printer. Draw the output as you think it would be printed. Items contained in square brackets are single ASCII characters.

    [FF] [ESC] – 1 My Favourite subject [ESC] – 0 [LF] [LF] [CR] Definitely Software Design and Development. [LF] [CR] [ESC] 4 It's the best! [ESC] 5 [LF] [LF] [CR] O [BS] X O [BS] + O [BS] X [FF]

14. Create a data stream of ASCII characters to produce the following output at the top of a page:



15. To be really annoying you wish to create a data stream for the printer that prints 'Gotcha' two lines down at the top of 100 sheets of paper. Create an algorithm you could use to create this data stream.



CHAPTER 10 REVIEW

1. The decimal equivalent of the binary number 10011110 is
   (A) 30
   (B) 159
   (C) 158
   (D) 94

2. CAB_{16} is equal to
   (A) 1111101011_2
   (B) 1110101011_2
   (C) 110010101011_2
   (D) 001111011011_2

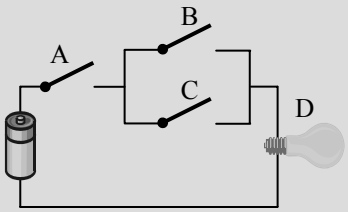
3. The word 'BIRD' coded in 7 bit binary ASCII is
   (A) 66, 73, 83, 68
   (B) 98, 105, 114, 100
   (C) 1000010, 1001001, 1010010, 1000100
   (D) 1100010, 1101001, 1110010, 1100100

4. Using 4 bit two's complement the range of integers that can be represented is
   (A) -16 to 15
   (B) -8 to 7
   (C) 0 to 15
   (D) -32 to 31

5. In terms of binary adders which of the following is true?
   (A) 8 half adders can be combined to create an 8 bit adder.
   (B) A half adder contains two full adders.
   (C) A full adder requires three inputs and two outputs.
   (D) Full adders do not take account of carries from previous additions.

6. Which of the following statements are true in regard to two's complement?
   (A) Addition can result in overflow errors.
   (B) Subtraction is performed by adding the negative.
   (C) You can always ignore carries in the final stage of an addition.
   (D) All of the above.

Use the diagram below to answer questions 7 and 8.



7. This diagram is best described as
   (A) AB+C
   (B) $A(B+C)$
   (C) $\overline{A(B+C)}$
   (D) A+BC

8. A truth table is constructed for the above circuit. If the inputs A, B and C are listed on the table in numerical order then the outputs at D would be
   (A) 11111000
   (B) 00000111
   (C) 01110111
   (D) 00001111

9. Flip-flops need to operate in sequence. The component that allows this to happen is the
   (A) edge trigger component.
   (B) clocking component.
   (C) latch component.
   (D) register component.

10. Which of the following is true of sensors and actuators?
    (A) Sensors perform mechanical tasks whereas actuators convert analogue data into digital.
    (B) Sensors convert input from the environment into digital form. Actuators perform mechanical output using data from the software.
    (C) Sensors react to the environment by providing feedback to actuators.
    (D) All input comes from sensors and all output goes to actuators.

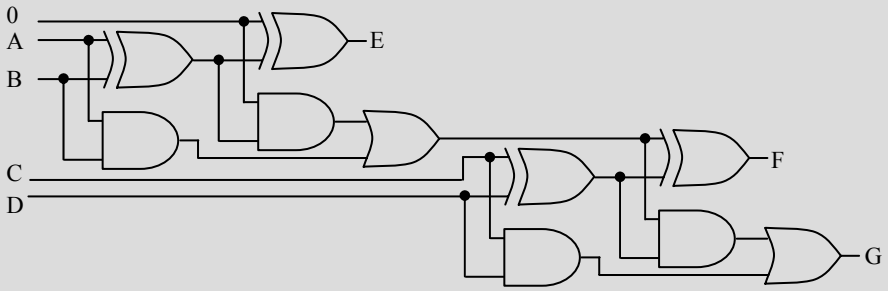
11. Perform the following binary calculations. Show all your working.

(a) 01010111 + 00011100	(b) 01011101 + 01110100	(c) 11011001 - 00110001	(d) 11000000 - 01110000
(e) 10111 × 11001	(f) 1011011 × 0001110	(g) 110110110 ÷ 1011	(h) 110111010 ÷ 110

12. Convert each of the following binary numbers into their 'IEEE single precision floating-point' equivalent.

    (a) 1101 (b) -1.101×2^{1011} (c) $-1.101101 \times 2^{-101}$ (d) 1.110101

Use the following circuit design to answer questions 13 and 14.



13. Copy and complete the following truth table for the above circuit design.

Inputs					Outputs		
0	A	B	C	D	E	F	G
0	0	0	0	0			
0	0	0	0	1			
0	0	0	1	0			
0	0	0	1	1			
0	0	1	0	0			
0	0	1	0	1			
0	0	1	1	0			
0	0	1	1	1			
0	1	0	0	0			
0	1	0	0	1			
0	1	0	1	0			
0	1	0	1	1			
0	1	1	0	0			
0	1	1	0	1			
0	1	1	1	0			
0	1	1	1	1			

14. What do you think is the purpose of the above circuit? What component circuits have been used to form this circuit design? Why is one of the inputs always zero?

15. Registers are used to store and shift data. What are the main component circuits contained in shift registers? How do these component circuits operate to store binary digits? Draw a circuit design that is able to store a binary digit. Use this design to explain your answer.

GLOSSARY

4GL	See 'Fourth Generation Language'
abstraction	The hiding of detail by the presentation of a more general instance. In the programming environment, an example of this is the use of a subroutine, rather than the inclusion of detailed code. In object-oriented programming abstraction is the process of designing classes so they are reduced to their necessary attributes and methods.
acceptance testing	Formal tests conducted to verify whether or not a system meets its requirements. Acceptance testing enables the client to determine whether or not to accept the new system as complete.
accumulator	One of the registers present within the CPU. It stores the result of the latest computation carried out by the CPU.
actuator	A device under the control of a signal from a controller that performs a mechanical action.
address	The number given to identify a particular storage location in the computer's memory.
alpha testing	Testing of the final solution by personnel within the software development company prior to the product's release.
agile	Quick thinking, co-ordinated, active and lean. Adapts well to changing situations.
AI	Abbreviation for Artificial Intelligence.
algorithm	A series of unambiguous instructions or procedural steps that will result in the solution to a specific problem within a finite time.
ALU	Abbreviation for arithmetic and logic unit.
argument	An independent variable or constant that gives to a command, instruction or function the information it needs to perform an operation. In programming, it is also the value of a parameter passed between a calling program and the called subprogram.
arithmetic and logic unit	The part of the central processing unit capable of mathematical and logical operations. Abbreviated to ALU.
array	A data structure in which a collection of data items of the same type can be treated as a single entity. Individual data items are identified by an index or subscript.
ASCII	Acronym for American Standard Code for Information Interchange. A seven-bit code for the binary representation of 128 characters within the computer.
assembler	A piece of software that changes a program written in assembly language into machine language.
assembly language	A low-level language that is written using a mnemonic code (eg LDA, STO) as opposed to binary digits. An assembly language is specific to the CPU used in the computer.
assignment	A process in which a value is allocated to a variable.
attribute	A defined property of an entity or object. The data held by the object.
authorship	The original creator of a work.
backup	A copy, on disk or tape, of software or data that is made by the user to guard against accidental loss.
Backus-Naur form	A metalanguage used to define the syntax of a programming language. Named after its two authors. Abbreviated to BNF.
backward chaining	The process of arriving at a conclusion from a stated set of conditions. Assumes that a particular solution is true and then asks questions to verify that the necessary conditions are present.
benchmark	A point of reference or standard from which quality or excellence is measured.

benchmarking	The process of objectively assessing and comparing the performance of a product against a benchmark.
beta testing	Testing of the final solution by a limited number of users outside the software development company using real world data and conditions.
binary digit	A digit in the binary number system.
binary number system	The number system in which the base is two. It uses the digits 0 and 1.
binary search	An algorithm used to search structured data. The data must be ordered in either ascending or descending order. The algorithm repeatedly divides the data to be searched into two parts, discards one part and searches the other until either the target is found or it is determined that the target is not present.
bistable device	A device that is able to hold or maintain a particular state. A bistable device can maintain two possible states.
bit	A contraction of binary digit. It is either a 0 or a 1.
black box	A term used to indicate any component of a computer-based system for which the user need not know the details of its internal operation.
black box testing	A technique used to test software solutions. The inputs and expected outputs are known however the processes occurring are unknown. Also referred to as functional testing.
BMP	Bitmap (image). Basic Multilingual Plane which is UTF-16 code plane 0 (Unicode).
BNF	See Backus-Naur form.
Boolean algebra	A set of rules, laws and theorems by which logical operations can be expressed and solved. Developed by George Boole (1815-64).
Boolean operator	An operator that acts upon Boolean variables and values. The list of these operators includes AND, OR and NOT.
Boolean variable	A variable that can only take values which are either true or false.
boundary	The delineation between a system and its environment.
breakpoints	A method used in software debugging to denote a point at which the program is to temporarily halt execution. The programmer can examine or change the contents of variables at this point and then resume execution if appropriate.
bubble sort	A form of sorting by exchanging pairs of elements that are out of order, in a sequence of passes through the data structure, until no such pairs exist. On each pass, one of the elements (the larger or smaller of each pair depending upon the criterion for exchanging) travels to an end of the data structure.
CASE tools	Computer Aided Software Engineering- a range of software that is used to assist the developer with a variety of tasks required as part of the development process.
central processing unit	Abbreviated to CPU. The CPU retrieves, decodes, interprets and executes instructions. The CPU, which is incorporated in an integrated circuit, can be described as comprising of the control unit, arithmetic and logic unit and temporary memory. The temporary memory can be described as consisting of registers and the accumulator.
checksum	Error checking bits transmitted at the end of a data block. The value of these bits is calculated by adding (or otherwise) the bits in the data packet.
chip	A shortened form of silicon chip; an alternative name for integrated circuit.
class	The definition of a category of objects when using the object-oriented paradigm. The class defines all the common attributes and methods of the different objects that belong to it.
clock speed	The rate at which the internal oscillator (the system clock) in the control unit of the CPU sends out electronic pulses to synchronise its operations. Measured in hertz (Hz).

code of conduct	A set of standards by which software developers agree to abide by.
compilation	The process of converting an entire source program written in a high-level language into machine language before executing any of the program.
computer word	The number of bits that are processed by the registers in the CPU at one time.
configuration management	Systematic control of changes to a software product, the purpose being to maintain software integrity, traceability and accountability during the life of the product.
constraint	A factor that limits the scope of a solution. The solution must comply with the constraint.
contract	An agreement between two or more parties that is enforceable by law.
control character	Character used as part of a data packet generally used for synchronisation purposes.
control function	That part of the CPU's task that manages the sequence of operations. It is carried out by the control unit.
control structure	One of three basic structures (sequence, selection and repetition) which control the logic flow of an algorithm or program.
control system	A system of sensors and actuators operating under the direction of a controller.
control unit	That part of the CPU that carries out selection and retrieval of instructions from storage in sequence, interprets them and initiates the required operation.
copyright	The sole legal right to produce or reproduce a literary, dramatic, musical or artistic work, now extended to include software.
COTS	See 'Custom off-the-shelf packages'
counterfeiting	Original software that has been copied and is then on-sold as legal copies.
CPU	See 'Central Processing Unit'
custom off-the-shelf packages	Packages sold commercially that perform tasks and can be customised by the developer. Also known as COTS.
data dictionary	A comprehensive description of each identifier used in a program. This commonly includes: name, number of characters (size), data type, number of decimal places (if applicable) and a description of the purpose of each variable.
data flow diagram	A diagram that shows the logical flow of data through an organisation (system) or subsystem.
data validation	A check on entered data to ensure that it is of the correct data type and that its value is sensible.
debugging	The process of finding and eliminating logic and syntax errors in a program or algorithm.
decimal	A number expressed using the base 10 system.
decompilation	The process of taking executable machine code and generating the equivalent assembler code, so that it is more easily understood by a human. This process is often necessary when the executable code needs to be modified and the programmer does not have access to the source code.
design specifications	Specifications that form the basis for planning and designing the solution.
desk checking	Checking the validity of an algorithm by working through the written version of the algorithm using test data.
DFDs	See 'Data Flow Diagram'
direct conversion	A type of conversion where the old system is removed and the new system implemented immediately. Also referred to as Direct Cut-over.
driver	A specially written routine that generates appropriate test data used to test a lower level module before the higher-level modules are completed.
driver (interface)	Provides the interface between a hardware device and an application.

EBCDIC	Acronym for extended binary coded decimal interchange code. An eight-bit code for the representation of characters within a computer.
EBNF	Abbreviation for extended Backus-Naur form.
effector	Synonym for actuator.
encapsulation	The isolation of an object from its environment, so that changes to an object's design can be made without affecting other parts of the system, as long as the interface to that object remains the same.
end user	The final person or persons who operate a software package.
end user development	A process in which an application is developed by users who have knowledge of a relevant software package and can customise it to meet their needs.
environment	The elements outside a computer system which affect that system.
equity	The concept of fairness and impartiality to which all persons should be entitled as a right.
ergonomics	The study of the relationship between people and their working environment.
ethics	A set of moral principles accepted by society as a standard against which individual behaviour can be judged.
expert system	A computer program based on various artificial intelligence techniques, performing a specialised task at the level of a human expert. It is often partitioned into a knowledge base, an inference engine, and (possibly) an explanation system.
expert system shell	A general purpose expert system containing reasoning mechanisms and a skeletal knowledge base into which a user can inject specialised knowledge.
extended Backus-Naur form	An extended and modified form of the BNF metalanguage for the definition of the syntax of a programming language. Abbreviated to EBNF.
feasibility study	A study that determines if the development should commence.
feedback	The return of some part of output to be used as input in a closed loop computer-controlled system.
fetch-execute cycle	The cycle of events that a computer carries out to obey an instruction. During the fetch part of the cycle, the address of the next instruction is used to access the memory and the contents of that location are placed into a register. During the execution part of the cycle, the instruction in the register is decoded and carried out.
file	A block of data comprising a related set of records that may be written to a storage device. May be made up of records, fields, words, bytes, characters or bits.
file association	Defines the application that is executed when a data file is opened.
fixed-point representation	The internal representation of numbers in which each number is stored as a single set of digits. The point that separates the integer part from the fraction part is in a fixed position within the set. Compared with floating-point representation, this method limits the range of numbers the computer can manipulate, but the operation is relatively faster.
flag	A variable used in a program (usually in the development or debugging stages) to indicate the result of a process, or that the process was executed.
Flip-flop	A circuit used in RAM chips as a memory store. Contains a latch, clocking and edge trigger component.
floating-point representation	The internal representation of numbers in which each number is stored as two sets of digits, one holding the mantissa and the other holding the exponent. The value of the exponent causes the decimal (binary) point to float until it rests in the final, correct place. Compared with fixed-point representation, this method allows a much wider range of numbers that the computer can manipulate, but the operation is relatively slower.
flowchart	A pictorial method of describing a system in which the various steps are represented by symbols.

forward chaining	The process of arriving at a conclusion from a stated set of conditions. Starts from the beginning of the facts and rules and asks questions to determine which path to follow next to arrive at a conclusion.
forward compatible	The ability of an older system to operate in harmony with a new system.
Fourth Generation Language	A type of programming language used in RAD.
freeware	Software that can be distributed freely as long as the distribution is not for profit.
full-adder	Used to add two binary digits together with a carry bit from a previous addition.
function	A predefined set of operations that returns a value. Built-in ones are those provided by the application software. In a programming language, it is a procedure (or subroutine) that returns a value. Most programming language implementations provide libraries of standard ones.
functional language	A type of programming language based on the functional paradigm. Essentially programs are constructed using functions. Examples include APL, Haskell and Lisp.
Gantt chart	A chart showing timelines for different stages of a systems development project.
Graphical User Interface	A method of presentation employed by a developer presenting the information (usually the screen design) in a graphical format.
GUI	See ' Graphical User Interface'
half-adder	Used to add two binary digits where no carry bit exists.
hardware	The physical units that make up a computer or any device working with the computer.
heuristic	A rule of thumb or a technique based on experience and for which our knowledge is incomplete. It works with useful regularity but not necessarily all the time.
hexadecimal	A number system that uses sixteen as its base. Because it requires sixteen symbols, the letters A to F are used as digits to represent the decimal numbers 10 to 15.
high-level language	A computer programming language in which each instruction or statement corresponds to several machine code instructions. It allows users to write in a notation with which they are relatively familiar.
IC	Abbreviation for integrated circuit.
imperative language	see procedural language.
incremental compilation	A method of translating high-level source code into object code. Modules that have not been altered, since the last compile, are not recompiled. Only altered modules are compiled. This translation method saves time when many developers are working simultaneously on aspects of a single project.
index	An integer value used to denote a particular data item held in an array. Also called a dimension or subscript.
inference engine	The section of an expert system program that contains the logic processing functions. Used to draw conclusions from stated facts and relevant rules.
inheritance	The ability of objects to take on the characteristics of their parent class or classes. Inheritance encourages modularity and robustness in code.
input process output diagram	Diagrams or charts that describe the data entering a process, the nature of the processing performed and the resulting information leaving the process.
insertion sort	A form of sorting by dividing the data structure into a sorted part and an unsorted part, taking each element in the unsorted part in turn, finding its correct position in the sorted part and inserting it there, thus increasing the size of the sorted part and decreasing the size of the unsorted part.
instantiation	The process of creating a particular object based on a class definition. Creating an ‘instance’ of an object.

integer	One of the (infinite) set of the positive and negative whole numbers including zero. In any computer application the range of allowable values is determined by the way in which the hardware represents an integer. Typical ranges are from -32768 to +32767 if the system uses 16 bits and two's complement.
integrated circuit	A circuit in which all the components are formed on a single piece of semi-conductor material (the silicon chip). Computers using them are said to be third generation, as compared with first generation machines using thermionic valves and second-generation machines using transistors.
intellectual property	Property resulting from the fruits of mental labour.
interpreter	A translator program that translates one line at a time of the high-level source code into a low-level object code, whereupon the computer executes that line before the next line is translated.
IPO	See 'Input Process Output diagrams'
knowledge base	A database used in artificial intelligence applications that consists of facts, inferences and rules needed to solve a problem.
knowledge engineering	The art of designing and building an expert system. In particular, collecting knowledge and heuristics from human experts in their area of speciality and assembling them into a knowledge base or expert system.
latch	A circuit that is able to store a binary digit. The output from two gates, provide input for each other.
lexical analysis	The process of examining each word to ensure it is a valid part of the language.
licence agreement	A signed agreement between the manufacturer and the purchaser of software that gives the purchaser the right to use the software.
linear search	An algorithm used to search structured data. The data need not be ordered. The algorithm compares each item of data with the target until either the target is found or it is determined that the target is not present.
linker	A system utility used to join compiled modules to form an executable program.
logic error	An error in the logic of the program that causes incorrect actions to be taken. Can normally only be detected by the programmer during the testing and debugging of the program when carefully selected test data is used as input to see that the program generates the known result.
logic gate	AND OR NOT NAND NOR XOR circuits or combinations of these circuits.
logical language	A programming language based on the logical paradigm. Facts and rules combine to form the knowledge base. An inference engine performs the processing. Examples include Prolog and expert system shells.
logical operand	An element that evaluates to one of the two values true or false.
logical operator	An operator that can be used with logical operands (eg AND, OR, NOT).
machine code/language	A language consisting of binary-coded instructions that can be understood and executed by the computer. A machine language is specific to the CPU of a particular computer.
macro	A term generally used to denote a routine, which is a set of instructions or a list of keyboard commands, stored in a file that can be executed by entering a few keys from the keyboard. They are used to save time and reduce the chance of keyboard errors in the execution of complicated or lengthy routines within an application.
marketing	The effort to determine and meet the needs and wants of current and potential customers.
metalanguage	A means of specifying the syntax of each of the valid commands in a given language.
method	The specification of a particular process to be performed on or by an object.
mischannelling	Software sold with discounted licences is resold as a fully licensed product

module	Performs a complete portion of a larger task and consists of one or more subprograms. It can pass data to and from other modules.
network	A system that allows a number of computers and their peripheral devices to be connected over a distance.
object	In an object oriented programming environment, this refers to the data structures and procedures that apply to a specific unit in the system.
object code	The machine code produced as a result of compiling a program written in a higher-level language (the source code); the code that runs when loaded into the computer.
object oriented language	A programming language based on the object oriented paradigm. Programs are made up of objects. Each object contains attributes and methods. Examples include C++, Java and Delphi.
one's complement	A method for representing integers in binary. Exchanging ones and zeros results in the one's complement.
operating system	The software that supervises and controls system resources such as the CPU, memory, storage devices and I/O devices; and executes other (user-controlled) programs. The operating system also manages the flow of data.
outsourcing	The contracting of services to external companies specialising in particular tasks.
overloading	Methods which have the same name but accept a different number of parameters or parameters of different types. Such methods which have the same name are said to be 'overloaded'.
overriding	And object-oriented programming concept where methods (or properties) replace an inherited method (or property).
packet	A group of binary digits containing data and control characters which form part of a complete digital message. The control characters permit the checking and correct re-assembling of the message.
paradigm	A philosophical or theoretical framework. A different approach under which laws, theories and generalisations are produced. Examples of programming paradigms include; imperative, logical, functional and object oriented.
parallel conversion	A type of conversion where the new system is introduced completely. For a period of time the old system remains operational.
parallel transmission	A transmission process in which a number of data bits are transmitted simultaneously, each bit along a separate channel or wire.
parameter	A value or variable passed to a function or procedure. It may be used to provide data to the function or procedure or to return data from it. In programming, this may be passed between procedures or modules.
parity bit	An additional bit attached to the binary code for each transmitted character or group of characters for the purpose of checking the accuracy of the received item.
patch	A piece of executable code that is inserted into an existing executable program to correct a problem identified in the software solution.
peer checking	A process whereby programmers not involved with the original design are asked to check the logic of an algorithm.
phased conversion	Aspects of the new system are progressively implemented until finally the new system completely replaces the old system.
pilot conversion	A method of conversion where a small amount of data is used with the new system to ensure it operates as intended. The new system is finally implemented in its entirety.
piracy	A euphemism for theft of intellectual property by the copying or use of software to which the user has no legal right.
plagiarism	The appropriation or imitating of another's ideas. The violation and infringement of a Copyright.

polymorphism	The concept that allows different objects to be used or presented in different ways at run time, depending on the users' requirements at the time.
procedural language	A programming language that requires programmers to specify a step-by-step process to tell the computer how to carry out a task, eg BASIC, Pascal, COBOL, FORTRAN, C. Based on variables and control structures. Also known as imperative languages.
procedure	A set of agreed actions to be carried out to complete a task.
processing	The process that manipulates data and information.
program	A sequence of instructions written in a programming language that directs a computer's actions.
prototype	An enactable model or mock-up of a software system that enables evaluation of features and functions in an operational scenario.
prototyping	Used to demonstrate to users/customers to ensure that the final program will meet their needs. It is often just a series of screens with no real processing taking place.
public domain	Software that is available due to the copyright holder relinquishing all rights to the software.
publisher	One whose business is the copying, distribution and sale of works by an author.
quality	A term that can be equated to the expectations of a customer. If expectations are met or exceeded then the product will be perceived as one of high quality.
quality assurance	A set of procedures used to certify that a generated product meets specified criteria with respect to quality and reliability. Quality assurance is ongoing throughout design and development.
RAD	See 'Rapid Application Development'
RAM	See 'Random Access Memory'
random access memory	Abbreviated to RAM. It is the primary memory, and is used to contain both data and instructions (programs) generated by the user. Data may be read from or written to this type of memory. Most technologies used for this memory make it a volatile memory.
rapid application development	A process in which a programmer makes use of software packages to quickly build applications to meet the users' needs.
read only memory	Abbreviated to ROM. Memory with instructions and/or data that are fixed at the time of production and cannot be erased or changed. ROM is non-volatile memory.
real number	One of the (infinite) set of numbers. In any computer application the range of allowable values is determined by the way in which the hardware represents a real number.
record	A collection of facts about an entity. A record comprises one or more related data items.
recursion	A function or procedure containing a call to itself. A form of repetition.
register	A group of fast access memory cells within the CPU. It is used for the temporary storage of data or instructions.
reliability	In the context of software design, it equates to an expectation that the software will perform its stated task efficiently and without error.
response time	The amount of time elapsing between the transmission of a command and the receipt of some response from the computer.
reverse engineering	The process of analysing an existing system to identify its components and their interrelationships, to allow the creation of a similar system.
ROM	See 'Read Only Memory'
royalty	Compensation, usually a portion of the proceeds paid to the author for the use of a work.

run-time error	An error that occurs when the object code is executed. It may be the result of a logic error, an undefined arithmetic operation such as division by zero, or an overflow error that occurs when a result is computed which cannot be correctly stored in the nominated memory location.
screen design	The design of the interface between the user and the software solution.
script	A list of commands that can be executed without user interaction.
selection sort	A repetition of a linear search to find the smallest (or largest) item in a list. Repeating the search on the remaining unsorted list leads to a sorted list.
sensor	Any device which converts input provided by the environment into a signal that can be communicated to a control program.
sentinel value	A dummy value used to signify the end of a data list, such as 'ZZZ' or 99999.
serial transmission	A form of data transmission in which the bits of each character are transmitted sequentially, one at a time, over a single channel or wire.
shared file	A file whose code is used by more than one program.
shareware	Software that is free of charge, however the manufacturer expects a voluntary monetary contribution to be made.
shift register	A register which allows bits to be moved left and/or right.
single-line stepping	The process of halting execution after each statement is executed. One of the tools used for the detection of errors in code.
start bit	A bit at the beginning of a character in asynchronous communication that signals the transmission of a character to the receiving computer.
stepwise refinement	An approach to problem-solving where the problem is broken down into successively smaller components until each individual component is easily solvable.
stop bit	A bit at the end of a character in asynchronous communication that signals the end of the transmission of a data stream to the receiver.
storyboard	An annotated sequence of rough drawings to represent a possible sequence in a proposed application, animation sequence or motion picture. The annotations provide general directions to the creator as to how the components should look and how each frame or screen relates to its predecessor and successor.
string	A term often used to describe a sequence of characters treated as a single entity.
structure diagram	A diagram used to model the hierarchy of processes within a system together with the sequence in which these processes take place.
structured approach	An approach to development that involves very structured, step-by-step stages. Each stage of the cycle must be completed before progressing to the next stage. Also known as the 'Waterfall Method'
stub	An incomplete procedure or function that has the same name and interface as the final form but is much simpler. Usually only contains an output statement to indicate that the procedure or function was called, together with any essential statements to ensure that related procedures and functions can work correctly. Used in testing the implementation of a program.
subprogram	A self-contained section of programming code that can be incorporated into a complete program. Can be entered from any point in the main program and usually returns to the instruction immediately following the branch to the subprogram. Have only one entry and exit point and are also known as procedures.
subroutine	Synonym for subprogram.
syntax	The rules governing the way in which elements of a programming language may be combined to form legal statements.
syntax error	An error resulting from an illegal statement in a programs code. The translator cannot continue once an error of this type is encountered.
temporary storage	Storage that is used to hold data for a relatively short time. It often uses a volatile storage medium.

top-down design	In programming, this implements stepwise refinement. An approach to program design that progressively breaks a large problem into a series of smaller, easier to solve problems. This leads to the creation of modules, thus reducing the programming complexity.
translator	A program that converts statements written in one programming language to the format of another programming language. Includes compilers, interpreters and assemblers.
truth table	A table for tabulating the outputs from a circuit design
two's complement	A method of representing integers in binary. The highest order bit is assigned a negative value.
unguarded loop	An alternative name for a repetition structure represented as a REPEAT-UNTIL construct in which the termination condition is tested after the loop body has been executed. The body of such a loop must therefore be executed at least once in a particular run of a program.
Unicode	A coding standard which specifies a unique number for every character in every language.
validation	A check carried out by the computer to ensure that only data conforming to certain rules is accepted for input to the program.
variable	The symbolic name of an addressed, stored entity in a program that is referred to for later processing.
virus	An unauthorised piece of programming code that causes unexpected results.
watch expression	An expression whose value is updated in the watch window as execution continues. One of the tools used for the detection of errors in code.
waterfall method	See 'Structured approach'
white box testing	A technique used to test software solutions. Explicit knowledge of the internal workings of the item being tested is used. Also referred to as structural or open box testing.
word	The number of bits that are processed by the registers in the CPU at one time.

INDEX

.NET 177
3G 95
4GL 68
802.11 95
abstraction 392
acceptance testing 314
acceptance tests 314
accumulator 250, 251, 253
accuracy 146
actuator 471, 476
address bus 253
agile 62, 63
agile alliance 65
agile approach 62
agreement 32
algorithm 151, 203, 206, 260, 263, 285, 346, 347
alpha testing 295, 302
amaya 53
AND gate 447
android market 49
angry birds 49
apple computer 53
apple's app store 49
applets 94
apps 94
arithmetic logic unit (ALU) 251, 253
arithmetic overflow 265
array 185, 202, 205, 206, 229
array of records 189, 228, 232
artificial intelligence (AI) 383
ASCII 192, 421, 422, 424, 425, 479
ASP 241
assembler 250, 252, 364, 439
atoms 372
attributes 367, 389
audience identification 215
author 11
axiomsys 79
back-to-base authentication 40
backup 262, 29, 33
backward chaining 375, 376
BASIC 177
basical multilingual plane 423
benchmark 304
benchmarking 221, 304
beta test 295, 351
bi-directional shift register 468
big O 153
binary 417
binary adders 458
binary addition 436
binary arithmetic 436
binary division 440
binary multiplication 439
binary number system 416
binary search 158
binary subtraction 438
binary to decimal conversion 418
binary to hexadecimal conversion 421
biometric devices 290
BIOS 264
bistable 462
bits 416
black box testing 296
bluetooth 95
BMP 423
boolean 168
boolean algebra 446, 451
boolean expressions 454
boolean variable 168
botnet 5
boundaries of the problem 108
breakpoints 271, 272
bubble sort 168, 356
bug 264
bus 253
business analysis 105
byte code 240
C++ 388
call 130
carry in 459
carry out 459
CASE tool 117, 130, 137, 139, 140, 205, 215, 285, 295, 302, 310, 326, 330, 333, 78, 92
CASE tools for testing 309
change management 333
changed requirements 320
changing nature of work 110
character representation 421
check boxes 211
child classes 391
chinese government 54
circuit design 449
circuit diagrams 454
class 390, 393
class declaration 390
class variables 403

client-server 45
clock 463
clocking component 463
closed loop system 476
cloud computing 95
code generation 247
code of conduct 17
code planes 425
code points 423
coding for variability 363
collaborative development 44
collaborative environments 95
combination boxes 211
command buttons 210
comments 283, 331
commercial 30
commercial systems 59
communication issues 119
communication with developers 312
communication with the client 312
comparison of actual with expected results 304
compatibility issues 107
compilation 242
compile 242, 247, 351
concept prototypes 140, 66
conceptual explanation 280
conceptual help 214
concurrent versions system (CVS) 92
configuration management 330
consistency in approach 216
constraints 112
constructor 393, 394
context diagram 125, 344
context sensitive help 214
contract programmers 91
control 130
control bus 253
control parameter 130
control structures 261
control systems 476
control unit (CU) 253
converting between hexadecimal, binary and decimal 418
copyright 111, 28, 29
correctness 145
cost effectiveness 112
COTS (customised off-the-shelf) 111, 220, 72
COTS package 218
COTS product 107, 298, 320, 326
counterfeiting 41
CPU 250, 252, 253, 362, 364
crash 264
cultural 21
custom-designed algorithms 201
custom-designed logic 185
customisation 111
customisation of existing software products 111
customised software 320
cyber safety 8
cyberbullying 8
data characters 473
data dictionary 137, 139, 346
data flow diagram (DFD) 126, 322, 64
data stream 472
data structure 185, 202, 261, 378
data type 138, 189
data validation 395
database management systems (DBMS) 363
datafactory 79
debugging output statements 267
decimal 417
decimal number system 416
decimal to binary conversion 419
decimal to hexidecimal conversion 420
declarative 383
decompilation 30
defining the problem 103
deleting 176
deMorgan's theorem 451, 453
derived classes 391
design specifications 117, 296
desk checking 268, 273
determining changes to be made 324
developer's perspective 117
developing a solution package 341
digital footprint 8
dimension 186
direct cut-over 80
disability 23
disk copy protection 38
division by zero 265
documentation 207, 278, 319
documenting changes 330
dominant developers of software 48
driver 266, 288, 332, 60
drop down lists 211
D-type flip-flop 464, 465
dynamic link libraries (DLLs) 247
EBNF 227, 244, 245
economic 22
edge trigger component 464

effects on level of employment 110
effects on the public 110
efficiency 145
efficiency of solution 408
electronic circuit 446
emerging technologies 289, 364
employment trends 89
encapsulated 395
encapsulation 389, 392, 393
encyption key 40
end user 72
end user development 72
end-user 118
ENIAC 362
environment 108, 226
ERG group 52
ergonomic issues 118
ergonomics 19
errors 263
evaluating 314
evaluation 295
event driven 226, 344, 388
event-driven 210
evolutionary prototypes 67
executable code 240, 242
expected output 270
expert system shell 384
expert systems 383, 384
exponent 432
extracting 175
facebook 7
fact 372
facts and rules 367
feedback 216
fetch-execute cycle 252
fields 189
file association 16
files 185, 192, 232
fixed-point 430, 431
flag 159, 251, 267
flip-flops 462
floating-point 430, 431
floating-point unit (FPU) 431
flowcharts 152, 261
format 138
forward chaining 375, 376
forward compatible 15
FTP 53
full adder 459
function 205
functional testing 309
functionality requirements 106
functions 152
gender 23
general fact 373
general purpose register 250
generating random numbers 177
GNU GPL 31
goal 373
google 54
GPS 95
graphical user interface (GUI) 219, 220, 226
grids 212
GUI 140, 53
half adder 458, 459
hardware 13
hardware environment 288
hardware serial numbers 38
head of a list 378
header information 473, 474
hearing disabilities 24
help screens 214
heuristic 384
hexadecimal 418
hexadecimal to binary conversion 421
hexadecimal to decimal conversion 418
HTML 107, 310, 93
human-computer interaction 290
hypertext 281
identity theft 9
IEEE 754 double precision floating-point standard 433
IEEE 754 single precision floating-point standard 432
image boxes 212
impact of software 3
imperative 366
imperative paradigm 362, 364, 365
implementation 225, 348
implementing 326
inclusivity 21
indenting of code 284
index 185
inference engine 374, 383
inheritance 391, 392
input 135
input and output data streams 471
input data stream 475
input process output (IPO) diagrams 135
inserting 176
insertion sort 162, 356
installation guide 278
instance variables 403

instantiated 393
instantiating 399
instantiation 390
instruction register (IR) 253
integer representation 427
integrity 145
intellectual property 11
interface 108
interface (or screen) design 118, 210
interfaces 298, 303
interfaces between modules 303
intermediate code 226
internet explorer 54
internet service provider (ISP) 108
interpretation 241
interpreter 247
interrupt 251, 251
interviews 105
intrinsic documentation 284, 331
IPO chart 323, 349
issues relevant to a proposed solution 109
java 177, 388
java virtual machine (JVM) 240
javascript 93
JK flip-flop 465
knowledge base 373
labels 212
large file sizes 299
latch 462
laws of boolean algebra 451
lead surrogate 425
learning content management systems (LCMS) 93
learning management systems (LMS) 93
learning objects 93
legal issues including licensing consideration 111
levels of testing 297
lexeme 243
lexical analysis 243, 244, 263
lexicon 243
liability 33
licence 32
licence terminology 32
licensing 32
limited prototype 345
limited use 33
linear search 155
linkedin 7
linker 247
list 378
list boxes 211
live test data 295, 298, 299
load testing 302, 309
logbooks 282, 342
logic 367
logic errors 263
logic gate 446, 449
logic paradigm 372, 409
machine code 242, 247, 250, 252
machine language 250, 364
macro 320
mainline 259
maintainability 146, 226, 319
maintenance 261, 285, 319, 351, 408
malware 18, 4
manifesto 65
mantissa 432
marketing 46
master-slave flip-flops 464
maximum and minimum values 156
meaningful identifiers 284
menus 210
metalanguages 227
metallica 55
methods 367, 389
methods of installation 80
microcode 250, 252
microsoft 53
microsoft office 48, 70
mind control devices 290
minimum hardware configuration 288
mischannelling 41
mix of transaction types 300
mobile phone technologies 95
model helicopter 479
modem 15
modular structure 259
modules 130
mosaic 53
mozilla 54
MSN 54
multi-dimensional arrays 185, 228
multiple inheritance 392
NAND gate 448, 449
napster 55
national privacy principles 26
navigation paths 141
navigator 54
needs of the client 103
netscape 53
network 44
network based software 45

neural networks 384
non-copyable datasheet 38
NOR gate 448
NOT gate 447
NSW Tcard system 52
object 390
object oriented 367, 388
object oriented languages 409
object oriented paradigm 388
objects 389
one logical task per subroutine 260
one's complement 428, 429
online help 281, 332
open control system 476
open office 48
open source 111, 112, 30, 92
operating system 14
operations 367
option buttons 212
OR gate 447
oracle designer 78
output 135
outsourcing 112, 89
overloading 395
overriding 397
ownership 32
PABX 473
paradigm 361, 371
parallel 81
parallel development 333
parallel in-parallel out shift register 467
parameter 130, 175, 205, 395
parent class 391, 397
parse tree 245, 247
parsing 245
pascal 228, 244
patch 319
pattern matching 383
peer checking 267
peer-to-peer (P2P) file sharing 55
performance issues 107
perl 93
personal information 25
personnel and procedures 298
phased 83
PHP 177, 241, 93
physical disabilities 24
picture boxes 212
pilot 84
piracy 28
place 47
plagiarism 28
polymorphism 398, 399
possible additional hardware 288
post implementation review 314
predicate 372
price 48
privacy 25
privacy act 1988 26
privacy issues 25
procedural explanation 280
procedural help 214
procedural language 390
procedural paradigm 362
procedures 152
process 130, 135
process diaries 282
processing strings 175
product 46
product of sums (PoS) 454
program counter (PC) 253
program traces 272
programmer productivity 406
prolog 372, 374, 377, 383
promotion 48
prototype 20, 345, 66
prototyping 66
PS2 ball mouse 471
PS2 port 472
pseudocode 152, 261
public domain 31
publisher 12
quality 12, 22
quality assurance 12, 145, 260, 295, 305
quality of solution 260
quantum computers 289
query 373
quick reference card 280
RACV versus Unisys 51
RAD (rapid application development) 45, 68
radio buttons 212
railroad diagrams 227, 245
random 232, 234
random access 194
random access file 194
random numbers 177
real numbers 430
records 189
recursion 265, 374
recursive call 265
reference manual 280
register 250, 253

registration code 39
relative 232
relative (or random access) files 194
reliability 146
reliance on software 6
reporting on the testing process 311
representation of data within the computer 416
representation of fractions 430
requirements 106
resetting variable contents 272
response time 301, 44
reusability 146, 260
reverse engineer 33
reverse/backwards engineering 30
rights and responsibilities of software developers 10
royalty 12
RS flip-flop 465
RS-NAND 463
RS-NAND latch 462
RS-NOR latch 462
rule 373
runtime errors 16, 264
scope 137, 138
screen designs 140, 141
script 320
scroll bars 212
scrolling electronic marquee 477
search engines 54
selecting an appropriate development approach 113
selection sort 165, 356
sensors 290, 471, 476
sentinel value 192
sequential 226, 232, 234
sequential files 192
serial in- serial out shift register 466
serial port 15
shared file 16
shareware 31
shift and add 439
shift and subtract 441
shift registers 466
sign and modulus 427
sign bit 432
single line stepping 272, 273
site licence 39
sixdegrees.com 7
social and ethical 110
social networking 6, 95
software development alliance 65
software development approach 59
software market 46
software piracy 36
software quality 145
software quality assurance (SQA) 145, 306
source code 240, 245, 270
source code documentation 283, 331
speciality circuits 458
speed of code generation 406
spyware 5
stack overflow errors 265
standard algorithms 153
step out 273
step over 273
storyboard 343
storyboards 141
structure chart 130, 201, 203, 259
structured approach 60
structured walk through 269
stubs 259, 266
subclass 397
subclasses 391
subroutine 130, 152, 206, 259
sum of products (SoP) 454
super class 391
surrogate values 425
surveys 105
swap 165
symbol table 244
syntactical analysis 245
syntax 245
syntax errors 263
system documentation 121
system flowcharts 122
system modelling tools 121
system models 117, 118, 285
system-level testing 297
systems analyst 105
table of symbols 243
tail of a list 378
Tcard 52
technical documentation 282
term 32
test data 207, 270, 295, 310
test driven development (TDD) 270
test results 312
testability 146
testing 326
testing and evaluating 295
text boxes 211
the copyright act 1968 29
the copyright amendment act 2006 29

theory of evolution 371
time management 105
token table 243
tool bars 211
top-down design 152, 260
touch sensing devices 290
tours 215
towers of Hanoi 366
trail surrogate 425
trailer information 473, 474
translation 240, 241, 243
trends in software development 92
trojan horse 5
truth table 447, 454
tutorials 215, 281
twitter 7
two's complement 428, 429, 438
two-dimensional arrays 186
unicode 192, 423
unique random numbers 178
unisys systems 51
UNL (unified modelling language) 64
usability 146, 20, 310
user defined predicate 372
user documentation 278, 351
user interface 19, 215, 216, 343, 345
user interface design 19
user manual 280
user story 64
user support 281
user's perspective 118
UTF-16 423, 425
UTF-32 423
UTF-8 423, 424
validation 138, 61
version control 262, 330, 333
version control systems (VCS) 262
virtual users 309
virtual world 95
virus 4
visual basic 218, 228, 344, 346, 70
visual disabilities 24
voice over IP (VoIP) 95
voice recognition 290
volume data 302
von Neumann architecture 362
W3C 107
warranty 32
watch expressions 273
waterfall method 60
web 2.0 tools 94
web browser 53
web-based software 93
white box testing 296
widgets 94
wizards 215
wordcloud 352
working in teams 341
world wide web consortium (W3C) 54
worms 5
XOR gate 448