VCE UNITS 3 & 4

software development

GARY BASS (LEAD AUTHOR)

SELINA DENNIS

THERESE KEANE

NELSON
A Cengage Company

Software Development
1st Edition
Gary Bass
Selina Dennis
Therese Keane
ISBN 9780170440943

Senior publisher: Eleanor Gregory
Project editor: Georgia O'Connor
Editor: Vanessa Lanaway
Indexer: Bruce Gillespie
Cover design: Chris Starr, MakeWork
Text design: Leigh Ashforth, Watershed Art & Design
Project designer: James Steer
Permissions researcher: Mira Fatin
Production controller: Karen Young
Typeset by: DiacriTech

Any URLs contained in this publication were checked for currency during the production process. Note, however, that the publisher cannot vouch for the ongoing currency of URLs.

Acknowledgements
Extracts from the VCE Applied Computing Study Design (2020–2023), are reproduced by permission, © VCAA. VCE is a registered trademark of the VCAA. The VCAA does not endorse or make any warranties regarding this study resource. Current VCE Study Designs, past VCE exams and related content can be accessed directly at www.vcaa.vic.edu.au

© 2019 Cengage Learning Australia Pty Limited

Copyright Notice
This Work is copyright. No part of this Work may be reproduced, stored in a retrieval system, or transmitted in any form or by any means without prior written permission of the Publisher. Except as permitted under the *Copyright Act 1968,* for example any fair dealing for the purposes of private study, research, criticism or review, subject to certain limitations. These limitations include: Restricting the copying to a maximum of one chapter or 10% of this book, whichever is greater; providing an appropriate notice and warning with the copies of the Work disseminated; taking all reasonable steps to limit access to these copies to people authorised to receive these copies; ensuring you hold the appropriate Licences issued by the Copyright Agency Limited ("CAL"), supply a remuneration notice to CAL and pay any required fees. For details of CAL licences and remuneration notices please contact CAL at Level 11, 66 Goulburn Street, Sydney NSW 2000, Tel: (02) 9394 7600, Fax: (02) 9394 7601
Email: info@copyright.com.au
Website: www.copyright.com.au

For product information and technology assistance,
in Australia call **1300 790 853**;
in New Zealand call **0800 449 725**

For permission to use material from this text or product, please email **aust.permissions@cengage.com**

ISBN 978 0 17 044094 3

Cengage Learning Australia
Level 7, 80 Dorcas Street
South Melbourne, Victoria Australia 3205

Cengage Learning New Zealand
Unit 4B Rosedale Office Park
331 Rosedale Road, Albany, North Shore 0632, NZ

For learning solutions, visit **cengage.com.au**

Printed in Singapore by 1010 Printing International Limited.
1 2 3 4 5 6 7 23 22 21 20 19

Contents

Preface v
About the authors vi
How to use this book vii
Outcomes ix
Problem-solving methodology xiii
Key concepts xvi

Unit 3

Introduction 1

Chapter 1 Introduction to programming 2

Data types 3
Data structures 5
Design brief 12
Representing designs 14
Files 17
Programming languages 21
Internal documentation 22
Naming conventions 24

Chapter 2 Development and features of a computer program 34

Processing features of a programming language 35
Algorithms for sorting 46
Algorithms for searching 53
Efficiency of algorithms 56
Validation techniques 57
Checking that modules meet design specifications 58
Preparing for Unit 3, Outcome 1 74

Chapter 3 Software analysis 75

What is a 'software solution'? 76
Project management 77
Collecting data 86
Functional and non-functional requirements 88
Software requirements specifications 91
Interfaces between solutions, users and networks 92
Security considerations 108
Next steps 112

Chapter 4 Software development: software design 122

Continuing Unit 3, Outcome 2 123
Software solution specifications 123
Generating design ideas 123
Evaluating design ideas 130
Evaluating the efficiency and effectiveness of solutions 132
Mock-ups 138
Preparing for Unit 3, Outcome 2 165

9780170440943

Unit 4

Introduction 167

Chapter 5 Software development and project evaluation 168

Managing files 169
Organising and manipulating data using data structures 174
Features of a programming language 182
Efficient and effective solutions 187
Techniques for checking coded solutions 188
Validation techniques 188
Testing 189
Recording the progress of projects 194
Factors influencing the effectiveness of the development model 198
Evaluating the efficiency and effectiveness of solutions and project plans 198
Next steps 202
Preparing for Unit 4, Outcome 1 209

Chapter 6 Cybersecurity risks 211

Physical and software security controls 212
Data security 215
Physical security 216
Software security 219
Software development practices 224
Strategies for minimising potential risks 229
Identifying software and data vulnerabilities 232
Strategies to protect against web application risks 237
Software acquired from third parties 244
Integrity of data 245
Next steps 251

Chapter 7 Software security 258

Why develop software? 259
Minimising risk 260
Key legislation for storage and disposal of data and information 260
Ethical issues 269
The impact of ineffective security strategies on data integrity 275
Data security 276
Preparing for Unit 4, Outcome 2 292

Index 293

9780170440943

Preface

This first edition of *Software Development VCE Units 3 & 4* is written to meet the requirements of the VCAA VCE Applied Computing Study Design that took effect from 2020.

This textbook looks at how a software development life cycle is implemented.

We believe that teachers and students require a text that focuses on the **Areas of Study** specified in the **Study Design**, and that presents information in a sequence that allows simple transition from theory into practical assessment tasks. We have therefore written this textbook so that a class can begin at chapter 1 and work their way systematically through to the end. Students will encounter material relating to the **key knowledge** dot points for each **Outcome** before they reach the special section that describes the Outcome. The Study Design outlines key skills that indicate how the knowledge can be applied to produce a solution to a client need or opportunity. These Outcome preparation sections occur regularly throughout the textbook, and flag an appropriate point in the student's development for each Outcome to be completed. The authors have covered all key knowledge dot points for the Outcomes from the Software Development VCE Units 3 & 4 course.

Our approach has been to focus on the key knowledge required for each school-assessed Outcome, and to ensure that students are well prepared for these; however, there is considerable duplication in the Study Design relating to the knowledge required for many of the Outcomes. We have found that, with an Outcomes approach, we are sometimes covering material several times. For example, knowledge of a problem-solving methodology is listed as key knowledge for five different Outcomes. In these cases, we have tried to provide general coverage in the first instance, and specifically apply the concept to a situation relevant to the related Outcome on subsequent encounters.

The authors allow teachers flexibility to develop the required key skills with their students within the context of the key knowledge addressed in this textbook and the resources available to them.

We have incorporated a margin column in the text that provides additional information and reinforcement of key concepts. The margin column also includes activities related to the topics covered in the text, and a consideration of issues relevant to the use of information systems.

Outcome features are included at several points in the book, indicating the nature of the tasks that students are to undertake in the completion of the school-assessed Outcome. The steps required to complete the Outcome are listed, together with advice and suggestions for approaching the task. The output and support material needed for submission are described. Sample tasks and further advice relating to the Outcomes are available at http://softwaredevelopment3and4.nelsonnet.com.au.

The chapters are organised to present the optimum amount of information in the most effective manner. The text is presented in concise, clearly identified sections to guide students through the text. Each chapter is organised into the sections described on pages vii–viii.

9780170440943

About the authors

Gary Bass teaches VCE Applied Computing at Year 11 and Year 12 in an online course environment at Virtual School Victoria. Previously he has taught VCE Physics, as well as developing and delivering middle-school ICT courses. Gary has presented at DLTV DigiCON and the annual IT teachers' conference on many topics, including Pop-up Makerspace; Big Data requires huge analysis – data visualisation; AR + VR = Mixed reality; and Marshall McLuhan – Medium is the message.

Selina Dennis is a Software Development and English Language teacher for the Department of Education and has been heavily involved in past and present Computing Study Designs. Selina has a Bachelor of Arts and Science in Computer Science and Linguistics from the University of Melbourne, and has a particular interest in Computational Linguistics. She spent several years in California in the computing industry as an engineering manager and director of engineering before entering teaching.

Associate Professor Therese Keane, Deputy Chair of the Department of Education at Swinburne University, has worked in a variety of school settings, where she has taught IT and led in K–12 education as the director of ICT. Her passion and many achievements in the ICT in Education and Robotics space have been acknowledged by her peers in her receiving national and state awards. Therese has presented numerous seminars and workshops for teachers involved in the teaching of IT. She has written several textbooks in all units of Senior IT in Victoria, VCE Information Technology since 1995. Therese's research interests include the use of technology in education, gender inequalities in STEM-based subjects, robotics in education and computers in schools for teaching and learning purposes. Therese is involved with the *FIRST* LEGO League the Championship Tournament Director for Victoria, and is a lead mentor for the RoboCats – an all-girl robotics team that participates in the *FIRST* Robotics Competition.

PUBLISHER ACKNOWLEDGEMENT

Eleanor Gregory sincerely thanks Gary, Selina and Therese for their perseverance and dedication in writing the manuscript for this book. She also thanks Tabitha Melgalvis for reviewing this book and providing valuable feedback to the authors.

9780170440943

How to use this book

KEY KNOWLEDGE

The key knowledge from the VCAA Software Development VCE Units 3 & 4 Study that you will cover in each chapter is listed on the first page of each chapter. The list includes key knowledge specified in the Outcome related to the chapter.

FOR THE STUDENT

The first page of each chapter includes an overview of the chapter's contents so that you are aware of the material you will encounter.

FOR THE TEACHER

This section is for your teacher and outlines how the chapter fits into the overall study of Software Development, and indicates how the material relates to the completion of Outcomes.

CHAPTERS

The major learning material that you will encounter in the chapter is presented as text, photographs and illustrations. The text describes in detail the theory associated with the stated Outcomes of the Software Development VCE Units 3 & 4 Study in easy-to-understand language. The photographs show hardware, software and other objects that have been described in the text. Illustrations are used to demonstrate concepts that are more easily explained in this manner.

Throughout the chapter, glossary terms are highlighted in bold, blue text, and you can find their definitions at the end of the chapter, in **Essential terms**.

The **School-Assessed Task Tracker** at the bottom of every odd-numbered page provides you with a visual reminder to help you track your progress in the school-assessed task so that you can complete all required stages on time.

MARGIN COLUMN

The margin column contains further explanations that support the main text, weblink icons, additional material outside the Study and cross-references to material covered elsewhere in the textbook. Issues relevant to Software Development that you can discuss with your classmates are also included in the form of 'Think about Software Development' boxes.

CHAPTER SUMMARY

The chapter summary at the end of each chapter is divided into two main parts to help you review each chapter.

Essential terms are the glossary terms that have been highlighted throughout the chapter.

Important facts are a list of summaries, ideas, processes and statements relevant to the chapter, in the order in which they occur in the chapter.

THINK ABOUT SOFTWARE DEVELOPMENT 3.1

Project-management tools are useful to find the perfect number of people needed on a task so it is finished as quickly as possible without anyone being idle. Using software, develop a Gantt chart to plan the baking of a cake. Assume you can use as many cooks as you want.

9780170440943

TEST YOUR KNOWLEDGE

These are short-answer questions that are provided to help you when reviewing the chapter material. The questions are grouped and identified with a section of the text to allow your teacher to direct appropriate questions based on material covered in class. Teachers will be able to access answers to these questions at http://softwaredevelopment3and4.nelsonnet.com.au.

APPLY YOUR KNOWLEDGE

Each chapter concludes with a set of questions requiring you to demonstrate that you can apply the theory from the chapter to more complex questions. The style of questions reflects what you can expect in the end-of-year examination. Teachers will be able to access suggested responses to these application questions at http://softwaredevelopment3and4.nelsonnet.com.au.

PREPARING FOR THE OUTCOMES

This section appears at points in the course where it is appropriate for you to complete an Outcome task. The information provided describes what the you need to do in the Outcome, the suggested steps to be followed in the completion of the Outcome and the material that needs to be submitted for assessment.

NELSONNET

The NelsonNet student website contains:

- multiple-choice quizzes for each chapter, mirroring the VCAA Unit 3 & 4 exam.
- additional material such as spreadsheets and infographics.

An open-access weblink page is also provided for all weblinks that appear in the margins throughout the textbook. This is accessible at http://softwaredevelopment3and4.nelsonnet.com.au.

The NelsonNet teacher website is accessible only to teachers and it contains:

- answers for the **Test your knowledge** and **Apply your knowledge** questions in the book
- sample SACs
- chapter tests
- practice exams.

Please note that complimentary access to NelsonNet and the NelsonNetBook is only available to teachers who use the accompanying student textbook as a core educational resource in their classroom. Contact your sales representative for information about access codes and conditions.

9780170440943

Outcomes

OUTCOME	KEY KNOWLEDGE	LOCATION
Unit 3 Area of Study 1 Outcome 1	**Software development: programming** On completion of this unit the student should be able to interpret teacher-provided solution requirements and designs, and apply a range of functions and techniques using a programming language to develop and test working software modules.	
Data and information	• characteristics of data types	p. 3
	• types of data structures, including associative arrays (or dictionaries or hash tables), one-dimensional arrays (single data type, integer index) and records (varying data types, field index)	p. 5
Approaches to problem solving	• methods for documenting a problem, need or opportunity	p. 12
	• methods for determining solution requirements, constraints and scope	p. 13
	• methods of representing designs, including data dictionaries, mock-ups, object descriptions and pseudocode	p. 14
	• formatting and structural characteristics of files, including delimited (CSV), plain text (TXT) and XML file formats	p. 17
	• a programming language as a method for developing working modules that meet specified needs	p. 21
	• naming conventions for solution elements	p. 24
	• processing features of a programming language, including classes, control structures, functions, instructions and methods	p. 35
	• algorithms for sorting, including selection sort and quick sort	p. 46
	• algorithms for binary and linear searching	p. 53
	• validation techniques, including existence checking, range checking and type checking	p. 57
	• techniques for checking that modules meet design specifications, including trace tables and construction of test data	p. 58
	• purposes and characteristics of internal documentation, including meaningful comments and syntax.	p. 22
Key skills	• interpret solution requirements and designs to develop working modules	p. 5
	• use a range of data types and data structures	p. 5
	• use and justify appropriate processing features of a programming language to develop working modules	p. 35
	• develop and apply suitable validation, testing and debugging techniques using appropriate test data	p. 58
	• document the functioning of modules and the use of processing features through internal documentation.	p. 22

9780170440943

OUTCOME	KEY KNOWLEDGE	LOCATION
Unit 3 Area of Study 2 Outcome 2	**Software development: analysis and design** On completion of this unit the student should be able to analyse and document a need or opportunity, justify the use of an appropriate development model, formulate a project plan, generate alternative design ideas and represent the preferred solution design for creating a software solution.	
Digital systems	• security considerations influencing the design of solutions, including authentication and data protection.	p. 108
Data and information	• techniques for collecting data to determine needs and requirements, including interviews, observation, reports and surveys.	p. 86
Approaches to problem solving	• functional and non-functional requirements	p. 88
	• constraints that influence solutions, including economic, legal, social, technical and usability	p. 91
	• factors that determine the scope of solutions	p. 90
	• features and purposes of software requirement specifications	p. 91
	• tools and techniques for depicting the interfaces between solutions, users and networks, including use case diagrams created using UML	p. 92
	• features of context diagrams and data flow diagrams	p. 96
	• techniques for generating design ideas	p. 123
	• criteria for evaluating the alternative design ideas and the efficiency and effectiveness of solutions	p. 130
	• methods of expressing software designs using data dictionaries, mock-ups, object descriptions and pseudocode	p. 134
	• factors influencing the design of solutions, including affordance, interoperability, marketability, security and usability	p. 140
	• characteristics of user experiences, including efficient and effective user interfaces	p. 141
	• development model approaches, including Agile, Spiral and Waterfall	p. 77
	• features of project management using Gantt charts, including the identification and sequencing of tasks, time allocation, dependencies, milestones and critical path.	p. 77
Interactions and impact	• goals and objectives of organisations and information systems	p. 149
	• key legal requirements relating to the ownership and privacy of data and information.	p. 152
Key skills	• select a range of methods to collect and interpret data for analysis	p. 86
	• select and justify the use of an appropriate development model	p. 144
	• apply analysis tools and techniques to determine solution requirements, constraints and scope	p. 91
	• document the analysis as a software requirements specification	p. 91
	• generate alternative design ideas	p. 123
	• develop evaluation criteria to select and justify preferred designs	p. 132
	• produce detailed designs using appropriate design methods and techniques	p. 134
	• create, monitor and modify project plans using software.	p. 80

9780170440943

OUTCOME	KEY KNOWLEDGE	LOCATION
Unit 4 Area of Study 1 Outcome 1	**Software development: development and evaluation** On completion of this unit the student should be able to develop and evaluate a software solution that meets requirements, evaluate the effectiveness of the development model and assess the effectiveness of the project plan.	
Digital systems	• procedures and techniques for handling and managing files and data, including archiving, backing up, disposing of files and data and security.	p. 169
Data and information	• ways in which storage medium, transmission technologies and organisation of files affect access to data	p. 173
	• uses of data structures to organise and manipulate data.	p. 174
Approaches to problem solving	• processing features of a programming language, including classes, control structures, functions, instructions and methods	p. 182
	• characteristics of efficient and effective solutions	p. 187
	• techniques for checking that coded solutions meet design specifications, including construction of test data	p. 188
	• validation techniques, including existence checking, range checking and type checking	p. 188
	• techniques for testing the usability of solutions and forms of documenting test results	p. 189
	• techniques for recording the progress of projects, including adjustments to tasks and timeframes, annotations and logs	p. 194
	• factors that influence the effectiveness of development models	p. 198
	• strategies for evaluating the efficiency and effectiveness of software solutions and assessing project plans.	p. 198
Key skills	• monitor, modify and annotate the project plan as necessary	p. 201
	• propose and implement procedures for managing data and files	p. 169
	• develop a software solution and write internal documentation	p. 22
	• select and apply data validation and testing techniques, making any necessary modifications	p. 188
	• prepare and conduct usability tests using appropriate techniques, capture results, and make any modifications to solutions	p. 192
	• apply evaluation criteria to evaluate the efficiency and effectiveness of the software solution	p. 198
	• evaluate the effectiveness of the selected development model	p. 199
	• assess the effectiveness of the project plan in managing the project.	p. 201
Unit 4 Area of Study 2 Outcome 2	**Cybersecurity: software security** On completion of this unit the student should be able to respond to a teacher-provided case study to examine the current software development security strategies of an organisation, identify the risks and the consequences of ineffective strategies and recommend a risk management plan to improve current security practices.	
Digital systems	• physical and software security controls used to protect software development practices and to protect software and data, including version control, user authentication, encryption and software updates	p. 212
	• software auditing and testing strategies to identify and minimise potential risks	p. 229

9780170440943

OUTCOME	KEY KNOWLEDGE	LOCATION
	• types of software security and data security vulnerabilities, including data breaches, man-in-the-middle attacks and social engineering, and the strategies to protect against these	p. 232
	• types of web application risks, including cross-site scripting and SQL injections	p. 237
	• managing risks posed by software acquired from third parties.	p. 244
Data and information	• characteristics of data that has integrity, including accuracy, authenticity, correctness, reasonableness, relevance and timeliness.	p. 245
Interactions and impacts	• reasons why individuals and organisations develop software, including meeting the goals and objectives of the organisation	p. 259
	• key legislation that affects how organisations control the collection, storage (including cloud storage) and communication of data: the *Copyright Act 1968*, the *Health Records Act 2001*, the *Privacy Act 1988* and the *Privacy and Data Protection Act 2014*	p. 260
	• ethical issues arising during the software development process and the use of a software solution	p. 269
	• criteria for evaluating the effectiveness of software development security strategies	p. 275
	• the impact of ineffective security strategies on data integrity	p. 275
	• risk management strategies to minimise security vulnerabilities to software development practices.	p. 276
Key skills	• analyse and discuss the current security controls to protect software development practices	p. 215
	• identify and discuss the potential risks to software and data security with the current security strategies	p. 218
	• propose and apply criteria to evaluate the effectiveness of the current security practices	p. 229
	• identify and discuss the possible legal and ethical consequences to an organisation for ineffective security practices	p. 275
	• recommend and justify an effective risk management plan to improve current security practices.	p. 276

Reproduced from the VCE Applied Computing Study Design (2020–2023) © VCAA; used with permission.

9780170440943

Problem-solving methodology

When an information problem exists, a structured problem-solving methodology is followed to ensure that the most appropriate solution is found and implemented. For the purpose of this course, the problem-solving methodology has four key stages: analysis, design, development and evaluation. Each of these stages can be further broken down into a common set of activities. Each unit may require you to examine a different set of problem-solving stages. It is critical for you to understand the problem-solving methodology because it underpins the entire VCE Applied Computing course.

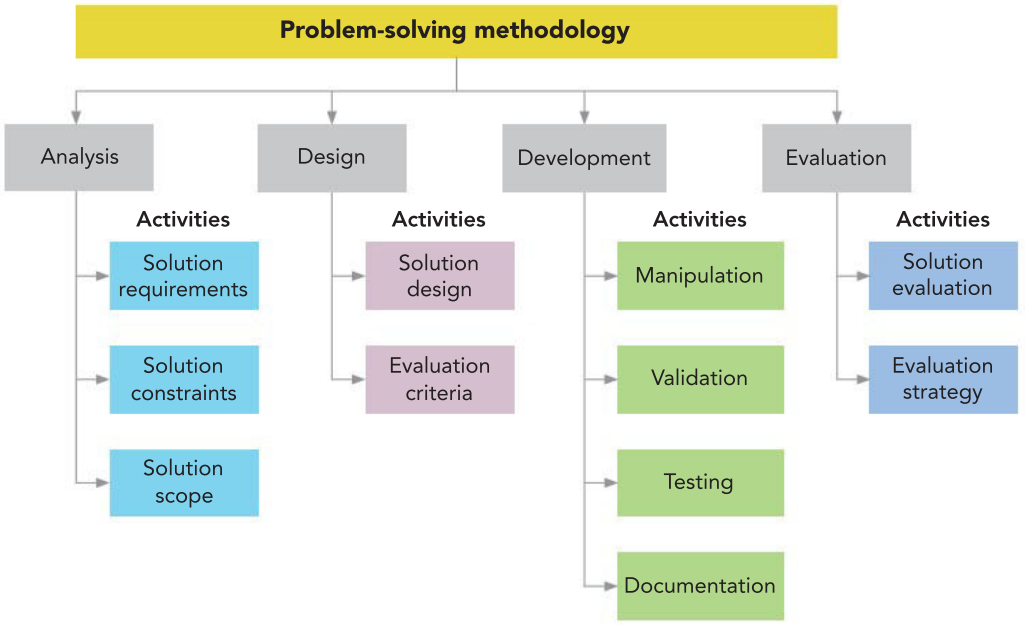


FIGURE 1 The four stages of the problem-solving methodology and their key activities

Reproduced from the VCE Applied Computing Study Design (2020–2023) © VCAA; used with permission.

Analyse the problem

The purpose of analysis is to establish the root cause of the problem, the specific information needs of the organisation involved, limitations on the problem and exactly what a possible solution would be expected to do (the scope). The three key activities are:

1. identifying solution requirements – attributes and functionality that the solution needs to include, information it must produce and data needed to produce this information
2. establishing solution constraints – the limitations on solution development that need to be considered. Constraints are classified as economic, technical, social, legal and related to usability.
3. defining the scope of the solution – what the solution will and will not be able to do.

9780170440943

Design the solution

During the design stage, several alternative design ideas based on both appearance and function are planned and the most appropriate of these is chosen. Criteria are also created to select the most appropriate ideas and to evaluate the solution's success once it has been implemented. The two key design activities are:

1. creating the solution design – it must clearly show a developer what the solution should look like, the specific data required and how its data elements should be structured, validated and manipulated. Tools typically used to represent data elements could include data dictionaries, data structure diagrams, input–process–output (IPO) charts, flowcharts, pseudocode and object descriptions. The following tools are also used to show the relationship between various components of the solution: storyboards, site maps, data flow diagrams, structure charts, hierarchy charts and context diagrams. Furthermore, the appearance of the solution, including elements like a user interface, reports, graphic representations or data visualisations, needs to be planned so that overall layout, fonts and their colours, for example, can be represented. Layout diagrams and annotated diagrams (or mock-ups) usually fulfil this requirement. A combination of tools from each of these categories will be selected to represent the overall solution design. Regardless of the visual or functional aspects of a solution design, at this stage a design for the tests to ultimately ensure the solution is functioning correctly must also be created.
2. specifying evaluation criteria – during the evaluation stage, the solution is assessed to establish how well it has met its intended objectives. The criteria for evaluation must be created during the design stage so that all personnel involved in the task are aware of the level of performance that ultimately will determine the success or otherwise of the solution. The criteria are based on the solution requirements identified in the analysis stage and are measured in terms of efficiency and effectiveness.

Develop the solution

The solution is created by the developers during this stage from the designs supplied to them. The 'coding' takes place, but also checking of input data (validation), testing that the solution works and the creation of user documentation. The four activities involved with development are:

1. manipulating or coding the solution – the designs are used to build the electronic solution. The coding will occur here and internal documentation will be included where necessary.
2. checking the accuracy of input data by way of validation – manual and electronic methods are used; for example, proofreading is a manual validation technique. Electronic validation involves using the solution itself to ensure that data is reasonable by checking for existence, data type and that it fits within the required range. Electronic validation, along with any other formulas, always needs to be tested to ensure that it works properly.

9780170440943

3 ensuring that a solution works through testing – each formula and function, not to mention validation and even the layout of elements on the screen, need to be tested. Standard testing procedures involve stating what tests will be conducted, identifying test data, stating the expected result, running the tests, stating the actual result and correcting any errors.

4 documentation allowing users to interact with (or use) the solution – while it can be printed, in many cases it is now designed to be viewed on screen. User documentation normally outlines procedures for operating the solution, as well as generates output (like reports) and basic troubleshooting.

Evaluate the solution

Some time after a solution has been in use by the end user or client, it needs to be assessed or evaluated to ensure that it has been successful and does actually meet the user's requirements. The two activities involved in evaluating a solution are:

1 evaluating the solution – providing feedback to the user about how well the solution meets their requirements, needs or opportunities in terms of efficiency and effectiveness. This is based on the findings of the data gathered at the beginning of the evaluation stage when compared with the evaluation criteria created during the design stage.

2 working out an evaluation strategy – creating a timeline for when various elements of the evaluation will occur and how and what data will be collected (because it must relate to the criteria created in the design stage).

9780170440943

Key concepts

Within each VCE Applied Computing subject there are four key concepts whose purpose is to organise course content into themes. These themes are intended to make it easier to teach and make connections between related concepts and to think about information problems. Key knowledge for each Area of Study is categorised into these key concepts, but not all concepts are covered by each Area of Study. The four key concepts are:

1. digital systems
2. data and information
3. approaches to problem solving
4. interactions and impact.

Digital systems focuses on how hardware and software operate in a technical sense. This also includes networks, applications, the internet and communication protocols. Information systems have digital systems as one of their parts. The other components of an information system are people, data and processes.

Data and information focuses on the acquisition, structure, representation and interpretation of data and information in order to elicit meaning or make deductions. This process needs to be completed in order to create solutions.

Approaches to problem solving focuses on thinking about problems, needs or opportunities and ways of creating solutions. Computational, design and systems thinking are the three key problem-solving approaches.

Interactions and impact focuses on relationships that exist between different information systems and how these relationships affect the achievement of organisational goals and objectives. Three types of relationships are considered:

1. how people interact with other people when collaborating or communicating with digital systems
2. how people interact with digital systems
3. how information systems interact with other information systems.

This theme also looks at the impact of these relationships on data and information needs, privacy and personal safety.

9780170440943

Unit 3

INTRODUCTION

In this study, software will be developed by applying the problem-solving methodology through the stages of analysis, design, development and evaluation.

In Unit 3 of Software Development, you will develop working software modules using a programming language (Unit 3, Outcome 1). You will then identify a suitable client, analyse a need or opportunity for that client, select an appropriate development model, prepare a project plan, develop a software requirements specification (SRS) and design a software solution. You will use all the stages of the problem-solving methodology (PSM) to prepare the project plan. This will complete the first half of the School-assessed Task (SAT) (Unit 3, Outcome 2). The second half of the SAT will be completed in Unit 4 (Unit 4, Outcome 1).

Area of Study 1 – Programming

OUTCOME 1 In this Outcome, you will respond to teacher-provided solution requirements and designs to develop working modules of a programming language. You will use a programming language to apply the problem-solving activities of manipulation (coding), validation, testing and providing documentation in the development stage.

Note: You will create a complete solution in Unit 4, Area of Study 1.

Area of Study 2 – Analysis and design

OUTCOME 2 In this Outcome, you will identify a software need or opportunity for a client. You will use the problem-solving methodology stages of analysis and design to complete part 1 of the School-assessed Task (SAT). Part 2 will be completed in Unit 4 (Unit 4, Outcome 1). You will use all the stages of the problem-solving methodology to prepare a project plan, using teacher-provided milestones.

A range of methods will be used to gather data for analysis. While a range of analysis tools, including use case diagrams, context diagrams and data flow diagrams will be used to describe the software solution to the need or opportunity. A full solution description is proposed using a software requirements specification (SRS), which entails documented details of requirements, constraints and scope of the solution. A design folio of several possible design ideas is generated, with a preferred design chosen by the client. This is then fully described with mock-up, pseudocode, object descriptions and data dictionary.

Contains extracts reproduced from the VCE Applied Computing Study Design (2020–2023) © VCAA; used with permission.

Shutterstock.com/whiteMocca

CHAPTER

1 Introduction to programming

KEY KNOWLEDGE

On completion of this chapter, you will be able to demonstrate knowledge of:

Data and information

- characteristics of data types
- types of data structures, including associative arrays (or dictionaries or hash tables), one-dimensional arrays (single-type data, integer index) and records (varying data types, field index).

Approaches to problem solving

- methods for documenting a problem, need or opportunity
- methods for determining solution requirements, constraints and scope
- methods of representing designs, including data dictionaries, mock-ups, object descriptions and pseudocode
- formatting and structural characteristics of files, including delimited (CSV), plain text (TXT) and XML file formats
- a programming language as a method for developing working modules that meet specified needs
- naming conventions for solution elements
- purposes and characteristics of internal documentation, including meaningful comments and syntax.

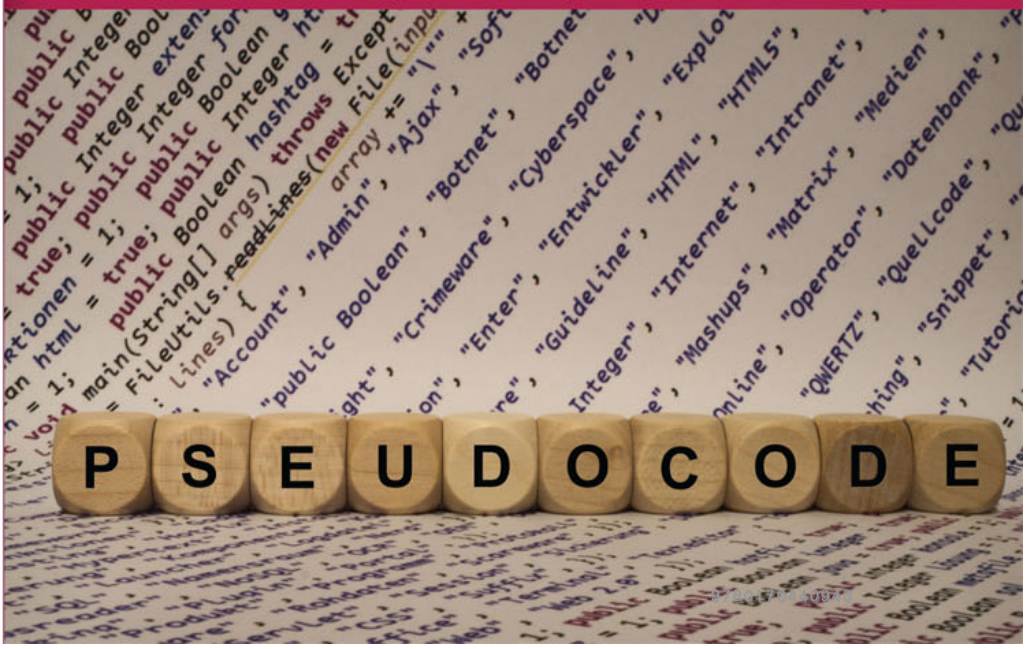
Reproduced from the VCE Applied Computing Study Design (2020–2023) © VCAA; used with permission.

FOR THE STUDENT

While programming languages differ, the fundamental components and logic needed to write using a programming language are the same. Data types and structures are consistent across many different programming languages, as are the conventions for naming solution elements, internal documentation and the formatting and structure of different file types. Similarly, methods of representing designs are also independent of any programming language; in fact, this is their strength.

FOR THE TEACHER

The focus of this chapter is the key elements of programming that are platform and language independent, as well as the methods used to create design briefs and represent designs. This chapter forms the basis of the background needed to prepare students for Unit 3, Outcome 1 as well as Unit 3, Outcome 2.



©Shutterstock.com/Dominik Bruhn

Data types

In programming, a **data type** is a method of classifying a **variable** to determine the data that variable can contain, as well as how the variable can be manipulated – that is, what it can do, and what can be done to it. While programming languages vary widely from each other, data types do not; they are consistent across all programming languages. When programming, it can be important to choose an appropriate data type when creating a variable. It is also important to select the most efficient data type. For example, it is not efficient to select a numeric data type that supports decimal places when creating a variable if that variable will only ever contain whole numbers. Similarly, storing a number as a string is not as efficient as storing it as a numeric data type, even if it is possible to convert strings to numbers.

Some languages, such as Perl, Ruby and Swift, are dynamically typed, which means that type checking for variables is completed when the program is run rather than in the code itself. Programmers do not need to set variable data types in dynamically typed languages.

Numeric

The **numeric** data type consists of whole numbers, referred to as **integers**, and decimal numbers, referred to as **floating points**. Integers can be referred to as *unsigned*, which means they can only store positive whole numbers, or *signed*, which means they can store both positive and negative whole numbers.

All numeric data types can have mathematical operations performed on them. The fundamental operations shown in Table 1.1 are the most common.

TABLE 1.1 Fundamental data type operations

Addition	+
Subtraction	–
Multiplication	*
Division	/
Whole number division (quotient)	/ /
Remainder after division (modulo)	%
Powers	**
Assign values	= or ←

When more than one operation appears within a line of code, the order of operations follows the same rules as BODMAS in mathematics: brackets, orders, division and multiplication, addition and subtraction. If two operators have the same precedence, they are evaluated from left to right.

Numeric data can also undergo comparisons, with the comparisons shown in Table 1.2 being the most common.

TABLE 1.2 Data type comparison operations

Less than	<
Less than or equal to (or ≤)	<=
Greater than	>
Greater than or equal to (or ≥)	>=
Equal to	== or =
Not equal to	!= or < >

SCHOOL-ASSESSED TASK TRACKER

☐ Project plan	☐ Justification	☐ Analysis	☐ Folio of alternative designs ideas	☐ Usability tests	☐ Evaluation and assessment	☐ Final submission

9780170440943

The power of 31 for signed integers represents 32 bits, minus 1 bit that is needed to determine if the signed integer is positive or negative. The 1 that is subtracted from the total is due to computer systems counting from 0, rather than 1.

THINK ABOUT SOFTWARE DEVELOPMENT 1.1

How big is 2^{64}? How big would the number be if we began using 128-bit computer systems?

THINK ABOUT SOFTWARE DEVELOPMENT 1.2

Research the likelihood of 128-bit computer systems becoming the norm within your lifetime. What would prevent these systems being used in standard computers?

The maximum and minimum values for floating point numbers are not as easy to determine as integers. When single-precision floating point numbers are stored, 1 bit is used for the sign (positive or negative), 8 bits for the exponent, and 23 bits to store the significant digits of the floating point number (that is, the fraction). For interest, the formulae that calculate the maximum and minimum for 32-bit systems and 64-bit systems can be found at the weblink.

32-bit systems
64-bit systems

Integer

Integers are commonly represented internally in a computer system as a group of binary digits, called *bits*. A bit is the smallest unit of data in a computer, and has a single binary value, 0 or 1. Bits are stored in multiples of eight, referred to as bytes; therefore, there are eight bits to a byte.

The maximum and minimum values of an integer depend on the computer architecture used to run the program and whether the integer is signed or unsigned.

In **32-bit computer systems** , integers that are signed have a minimum value of -2^{31} and a maximum value of $2^{31} - 1$; from –2147483647 to 2147483647. Unsigned integers have a minimum of 0 and a maximum of $2^{32} - 1$ (4294967295).

In **64-bit computer systems**, integers that are signed have a minimum value of -2^{63} and a maximum value of $2^{63} - 1$. Unsigned integers have a minimum of 0 and a maximum of $2^{64} - 1$.

It is important to know the computer architecture on which a program will run before designing and developing a software solution. Going beyond the maximum and minimum values of integers can result in an **integer overflow**, which may result in a program crashing, or producing inconsistent or invalid output. Integer overflows compromise a software solution's reliability and security.

Floating point

Floating point numbers, also referred to as 'floats' or 'doubles', are the computer representation of real numbers; that is, numbers that allow for decimal places.

Floating points consist of two main parts:

- a *significand*, which contains the digits of the number that is represented. These can be either positive or negative.
- an *exponent*, which helps determine where the decimal point is placed within the significand.

Two basic formats of a floating point number in computer systems are single precision and double precision. Single precision is used in 32-bit systems and double precision in 64-bit systems.

Character

The **character** data type is a symbol that has meaning. It can consist of any single meaningful unit, such as a letter, a number, a punctuation mark, a symbol, or even a space. For example, the word 'example' has 7 characters.

What is determined as 'meaningful' relies on something referred to as **character encoding**. Character encoding is a way that a computer program can translate binary data into meaningful characters. There are many character-encoding schemes that handle different character sets. For example, ASCII is a character-encoding scheme that represents English characters, punctuation and numbers. UTF-8 encoding is a character-encoding scheme that can represent characters from other languages, such as Japanese Kanji and Korean Hanja, as well as symbols such as those representing the euro (€) and yen (¥).

A set or sequence of characters is referred to as a **string**. For example, the string 'I like pie' consists of 8 characters that are letters, and two characters that are spaces. Strings are often implemented in programming languages as an array of characters.

9780170440943

Characters and strings can undergo the same comparisons as numeric data types, as listed in Table 1.2. Depending on the programming language used, they can also undergo some of the fundamental data type operations listed in Table 1.1, such as addition, multiplication and assignment. An example of string addition can be seen in Figure 1.1.

```
INPUT firstName
PRINT "Hello " + firstName + "!"
```

FIGURE 1.1 Pseudocode example of string addition

Boolean

Boolean data types have only two possible values: 0 and 1. In a programming language these are often referenced with the words 'False' or 'True', respectively. This data type is named after George Boole, a 19th-century mathematician who was the first to define an algebraic system of logic. Boolean data types are very useful for systems that require decisions to be made or conditions to be met.

Much like integers, boolean values can be treated mathematically, allowing for comparison operators such as those listed in Table 1.2. This allows for boolean logic operations to occur in any programming language.

Boolean values can also be used with the fundamental operators *and*, *or* and *not* in statements where a condition must be met. For example, if a program is required to turn on a light in a room if it is dark and the light is not already on, it could use two boolean values to test for this condition (Figure 1.2).

```
IF isDark = True AND lightOn = False THEN
    turnOnLight()
ENDIF
```

FIGURE 1.2 Pseudocode example of boolean test conditions

Data structures

A **data structure** is a method of organising data to allow particular operations to be performed on them efficiently; in this way, they are more complex than data types. The types of data structures used in Software Development are: arrays, associative arrays such as hash tables and dictionaries, stacks, queues, linked lists, files, fields, records and classes.

Array

An **array** is a data structure that contains groupings of data. These elements are traditionally of the same data type, such as character, numeric or boolean. Arrays can also store groupings of other data structures, such as fields, records, or even other arrays. Arrays are very useful in programming, as they allow for related sets of data to be organised and ordered efficiently.

1.3 THINK ABOUT SOFTWARE DEVELOPMENT

You may come across the term 'null terminated string' when looking at programming language reference manuals and computer science texts. What is a null terminated string?

Boolean values take up only a small amount of space in memory, so it is tempting to use them to store any type of data that seems to only have two values. However, it is important to consider future expansions to programs before making a decision that will limit a data type. For example, many old systems that had gender stored as a boolean value are now being rewritten to change gender to a character or string data type.

Some programming languages, such as Python, allow arrays to contain more than one data type.

SCHOOL-ASSESSED TASK TRACKER

☐ Project plan | ☐ Justification | ☐ Analysis | ☐ Folio of alternative designs ideas | ☐ Usability tests | ☐ Evaluation and assessment | ☐ Final submission

9780170440943

For example, a teacher might collect data related to the height of each of their students and store this data as a one-dimensional array of floating point numbers. This is a much more efficient method of storing data than creating separate variables to store each student's height, as it allows for faster sorting and searching.

The contents of an array are referenced using an index value – often an integer starting at 0. The way that an array is stored means that each element has a set position within it. This allows for quick access to a particular element of the array, without necessarily needing to check every element.

Arrays can typically use the operations shown in Table 1.3.

TABLE 1.3 Array data structure operations

Add or append	+
Remove or delete	–
Lookup	arrayName[indexValue]

```
arrayHeights ← [ 1.23, 1.35, 1.21, 1.61 ]
firstStudent ← arrayHeights[0]
secondStudent ← arrayHeights[1]
fourthStudent ← arrayHeights[3]
```

FIGURE 1.3 Pseudocode example of an array

Consider the array in Figure 1.3. In this example, `firstStudent` would contain the floating point number 1.23, `secondStudent` would contain 1.35 and `fourthStudent` 1.61.

In pseudocode (covered later in this chapter), arrays are sometimes indexed starting at **1**, but this should always be made clear in the pseudocode comments.

Associative array

An **associative array** is a special type of array data structure that consists of a collection of key and value pairs, where the key is unique and can be of any data type or structure. This makes it more flexible than an array with an integer index.

Associative arrays can typically use the operations shown in Table 1.4.

For example, if a teacher were to collect student heights in a one-dimensional array, they would not be able to go back and find a particular student's height, as that data (the student's name) was not stored in the array. If the teacher used an associative array, they could store the data they collected as (key, value) pairs, where the key is the student's name, and the value is the height of that student. This would allow the teacher to look up any particular student to find out their height.

TABLE 1.4 Associative array data structure operations

Add or append	assocArray[key].add (value) or assocArray[key] ← value
Remove or delete	assocArray[key].remove (value)
Modify or change	assocArray[key].change (value) or assocArray[key] ← value
Lookup	assocArray[key]

9780170440943

```
assocArrayHeights ← {}
assocArrayHeights["Paulo"] ← 1.23
assocArrayHeights["Shehara"] ← 1.35
assocArrayHeights["Phoebe"] ← 1.21
assocArrayHeights["Tuan"] ← 1.61

PRINT assocArrayHeights["Phoebe"]
```

FIGURE 1.4 Pseudocode example of an associative array

In the pseudocode example shown in Figure 1.4, Phoebe's height would be printed once the code was executed.

One important limitation of an associative array is that keys are not organised or sorted in a consistent way. If the teacher wanted to print the heights of every student in sorted order, this would not be possible using just an associative array.

To print the heights of every student in sorted order from an associative array, the teacher would need to extract the key elements of the associative array into a normal array and sort that array of key elements. They could then retrieve each value based on the sorted array of key elements.

Dictionary

A **dictionary** is a synonym for an associative array. Associative arrays can also be referred to as maps or symbol tables.

Hash table

A **hash table** is a particular type of associative array which, instead of (key, value) pairs, uses (key, bucket) pairs, where the *bucket* (or *slot*) is a one-dimensional array. The *key* in the (key, bucket) index is computed using a **hash function** on the *value* that is to be inserted. Once this key has been computed, the value is then inserted into the bucket at the correct position.

The benefit of using a hash table over a regular associative array is that it allows for efficient searching. When dealing with very large amounts of data, it can take a very long time to search through and find a particular element if every element has to be checked. Hash tables are faster, as they move elements into smaller array 'buckets', requiring fewer items to look through when searching. The better the hash function, the faster the search, with a **perfect hash** function resulting in a hash table that has only one element in each bucket. This is quite rare, however, and **imperfect hash** functions are far more likely. An imperfect hash function is a function that possibly computes the same *key* index for more than one value. This results in a **collision**, which must be handled within a software program. Collisions are often handled by inserting all matching values from the hash into an array attached to the key index.

Consider Figure 1.5, a simple hash table that contains words from a book, where the hash function looks at the first character of the value to obtain the key.

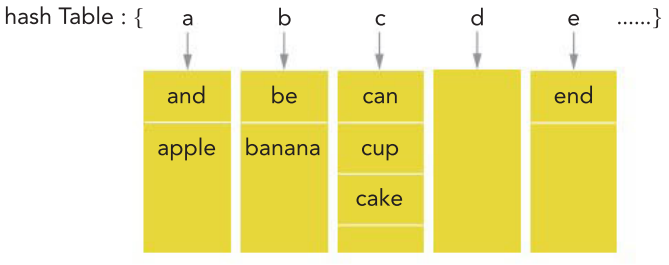


FIGURE 1.5 A hash table containing words from a book

SCHOOL-ASSESSED TASK TRACKER

☐ Project plan	☐ Justification	☐ Analysis	☐ Folio of alternative designs ideas	☐ Usability tests	☐ Evaluation and assessment	☐ Final submission

9780170440943

If the next word to be inserted into the hash table is 'durian', the hash function would return the key as 'd' and the value of 'durian' would be inserted into the bucket with the 'd' key, as shown in Figure 1.6.

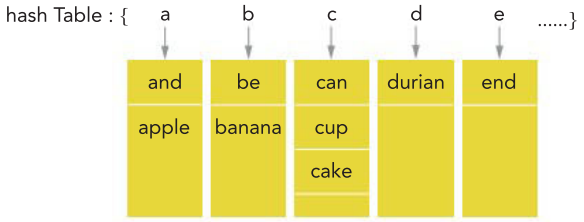


FIGURE 1.6 An updated hash table, with 'durian' added

If the next word after that is 'egg', the hash function would return the key as 'e'. A collision will occur, as the 'e' bucket already has 'end' in it, so the value 'egg' would need to be added to the end of the 'e' bucket. This is shown in Figure 1.7.

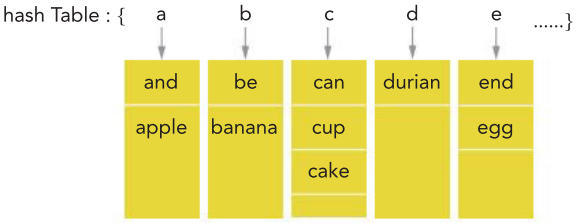


FIGURE 1.7 An updated hash table, with 'egg' added

Each time a collision occurs, the new value is added to the end of the bucket – notice that the array inside each bucket is not sorted.

Hash tables are most useful when there is a lot of data to store and none of it needs to be sorted. They are particularly useful for searching, as long as the hashing function does not create too many collisions.

Queue

A **queue** is a data structure that is best described using the analogy of a line at a cafeteria. The person at the start of the line is the next person served, and any new person joining the line adds themselves to the end of the line.

Much like queueing for food, elements of data in a queue are inserted at the end of the queue (**enqueue**), and each element can only be accessed by taking it from the start of the queue (**dequeue**). This is referred to as **first in first out (FIFO)** access.

©Shutterstock.com/Vibrant Image Studio

FIGURE 1.8 People in a queue

9780170440943

TABLE 1.5 Queue data structure operations

enqueue	queue.enqueue (value) inserts a value at the end of a queue
dequeue	queue.dequeue (value) removes and returns a value from the front of the queue
front	queue.front () returns a value from the front of the queue, without removing it
empty	queue.empty () returns a boolean value of true if the queue is empty, false if not
size	queue.size () returns the number of elements in the queue

Queues can contain any data type or structure, including other queues. In many programming languages, queues are implemented as arrays or linked lists. They are useful for implementing functionality to manage wait lists and access to shared resources (e.g. print queues), and for handling multiprocessing software with parallel processing needs.

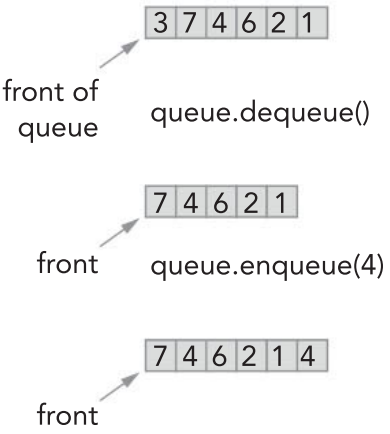


FIGURE 1.9 Queue operations on data using an array

Stack

A **stack** is a data structure that is most often described using the analogy of a stack of dirty dinner plates. As each plate is washed, it is removed from the top of the stack. The dirty plate that was underneath it is now at the top of the stack. Any new dirty plate is placed on top of the current stack of dirty plates.

©Getty Images/The Image Bank/Paul Taylor

FIGURE 1.10 A stack of dirty plates

Just like in the plates analogy, elements of data in a stack are inserted at the top of the stack (**push**) and each element can only be accessed by taking it from the top of the stack (**pop**). This is referred to as **last in first out (LIFO)** access.

SCHOOL-ASSESSED TASK TRACKER

☐ Project plan	☐ Justification	☐ Analysis	☐ Folio of alternative designs ideas	☐ Usability tests	☐ Evaluation and assessment	☐ Final submission

9780170440943

TABLE 1.6 Stack data structure operations

push	stack.push (value) inserts an element on top of the stack, pushing all other elements down one place in the stack
pop	stack.pop (value) removes and returns the last inserted element
top	stack.top () returns the last inserted element, without removing it
empty	stack.empty () returns a boolean value of true if the stack is empty, false if not
size	stack.size () returns the number of elements in the stack

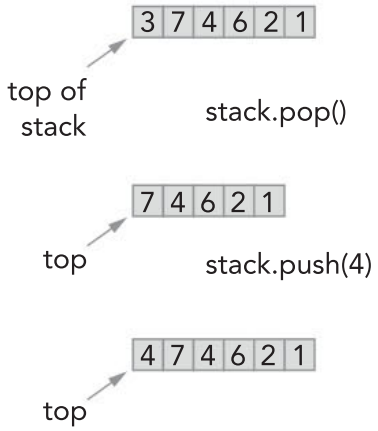


FIGURE 1.11 Stack operations on data using an array

Stacks can contain any data type or structure, including other stacks. In many programming languages, queues are implemented as arrays or linked lists. They are useful for implementing functionality such as an undo operation in word processing software, or for storing the history of visited web pages.

Linked list

A **linked list** is an ordered set of elements in which each element is connected to the next element in the list. This data structure allows data elements to be ordered into a sequence, and allows for efficient insertion and removal of elements from any position in the sequence. Linked lists are particularly useful in sorting algorithms.

In a linked list, each element is referred to as a **node**. Each node contains a data element as well as the memory address of the next node in the linked list. This is typically referred to as a **pointer**.

Linked lists are often used to implement stacks, queues and associative arrays. The simplest type of linked list is referred to as a *singly linked list*, which only has the ability to traverse the list in one direction. Another common type of linked list is a *doubly linked list*, which allows for two-directional traversal, as each element in the linked list keeps track of the next element as well as the previous one.

Linked lists make it very easy to add and remove elements at the start, middle and end of the list. Unlike arrays, whose index values must be shifted (increased) to make room for a new element to be inserted, linked lists only need to change the pointer to the next element in the list, and the pointer to the previous element if it is a doubly linked list.

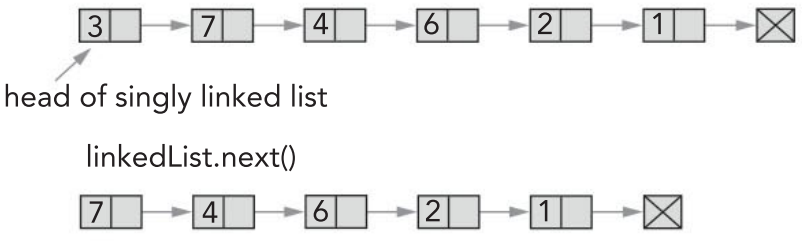


FIGURE 1.12 An example of a singly linked list of integers

TABLE 1.7 Data structure operations on a doubly linked list

Head	linkedList.head () returns the first element in the linked list
Tail	linkedList.tail () returns the last element in the linked list
Next	linkedList.next () returns the next element in the linked list based on the current element
Previous	linkedList.prev () returns the previous element in the linked list based on the current element

Record and field

A **record** is a basic data structure for collections of related elements. These elements may or may not be of the same data type. Most frequently, records are used in database systems but they are also commonly used in programming languages, where they are referred to as **structs**.

A record consists of a number of **fields** that are typically fixed – that is, the fields do not tend to change once the record is defined and used. Each field has a name and each has its own data type.

For example, a customer record may contain fields such as `firstName`, `lastName` and `dateOfBirth`.

Records are most useful when a collection of variables are related to each other. This provides a logical method of ordering data within a program so that data can be accessed quickly.

In object-oriented programming languages, a record is essentially an object that has no object-oriented features, containing only collections of fields and values. Records and fields can also exist in some types of structured plain text files.

As records contain programmer-defined fields, there are no set operations that can be listed for the record data structure. Rather, there are common operations that can be performed on the record and the fields within it, such as assignment and comparison, as well as adding or removing fields.

Class

Imagine a developer is writing a game where players play golf.

Alamy Stock Photo / ArcadeImages

FIGURE 1.13 An example of a golf game

SCHOOL-ASSESSED TASK TRACKER

☐ Project plan	☐ Justification	☐ Analysis	☐ Folio of alternative designs ideas	☐ Usability tests	☐ Evaluation and assessment	☐ Final submission

9780170440943

THINK ABOUT SOFTWARE DEVELOPMENT 1.4

Think about gaming in general. When would it be useful to keep statistics (variables) about something?

THINK ABOUT SOFTWARE DEVELOPMENT 1.5

In a gaming context, consider the speed of processing required to open and close files every time a player gains a stat or skill, or has damage dealt to them.

How many people played *Fortnite* concurrently at the peak of its popularity?

Imagine that many read/write operations – this would be an absurd amount of load on a system.

Describe other contexts where read/write operations would be an important factor to consider when writing software.

Before object-oriented programming introduced classes and objects, there was no straightforward way to write the golf clubs in the golf game with a programming language that just used functions, subroutines and variables. It would only be possible if the program saved information constantly to files and then read them back for each golf club and each player. Reading and writing to files is very slow and the game created could not be played by more than a couple players at a time.

In the game, the golf clubs that a player uses influence the outcome of the game. Players can upgrade golf clubs as they gain experience in playing courses. Each club carries information (variables) unique to that club, such as the quality of the club (its condition), the material of the grip and the material of the club itself.

A golf club also has actions associated with it, such as the action of swinging the golf club, or the action of breaking the golf club because a player is frustrated. For those actions, a programmer would need to know the quality, grip material and club material of each and any golf club used by a player.

What happens if each time a player uses the golf club, the quality decreases, the grip on the handle wears away a little, a new scratch is formed on the club and the player gets a little more frustrated?

Writing a function in the source code of the game would not easily help a programmer keep track of all those things for each golf club being used, particularly if there are many players and many golf clubs. The source code would need to have a different variable for each club a player has, as well as a different variable for each player who plays the game. This becomes quite tedious to maintain within code, because to avoid **hard-coding** variables containing players and their clubs, the code would need to have used multiple associative arrays that were synchronised with each other so that each index value matched a particular player.

Using a data structure called a **class** allows a programmer to solve the 'golf club' problem. A class is a programmer-defined data structure that exists in object-oriented programming languages. Classes group conceptually similar functions and variables together in one place and work as templates for creating **objects**, which are **instantiations**, or instances, of classes that exist in memory on the computer where the program is run. A useful analogy to describe a class is to think of it as a blueprint or architectural design, such as one used when building a house. It describes everything that needs to be built to make a house, but is not a house itself. The house that is built using the design is an instance of the house blueprint. Classes work in the same way. They describe all of the elements and components that are required by the object created from the class blueprint, and this object is referred to as an instance of the class. The number of objects that can be created from a class is limited only by the amount of memory (**RAM**) in the computer system on which the program is running.

The purpose of a class is to create a template for objects with pre-determined variables and behaviour. These templates can then be instantiated as objects or be used by another class in order to extend upon or change their behaviour. This allows for code re-use in programs where objects are similar to each other. In the golf club example, a programmer could extend on a base golf club class using **inheritance** to make woods, irons, wedges and putters.

Classes contain relevant variables, data types, data structures, methods and events. These are explained further in chapter 2.

Design brief

A design brief is a document or statement that outlines the nature of a problem, opportunity or need. It briefly describes processes, systems and users as a way to 'kickstart' the design process. The design brief is created during the analysis stage of the problem-solving methodology.

A design brief contains an overview of the solution requirements, any known constraints and a short discussion of the scope. Once the analysis stage of the problem-solving methodology has been completed, these elements are further developed and incorporated into a software requirements specification as part of the design stage of the problem-solving methodology, as outlined in chapter 2.

9780170440943

Solution requirements

Solution requirements are what the client needs from the solution. What features do they want in the solution? Solution requirements in a design brief are often worded descriptions, rather than technical descriptions.

Constraints

Solution **constraints** are factors that may limit or restrict solution requirements. At the stage when a design brief is created, these constraints are described only in general terms. Typically, constraints involve economic, technical, social, legal and usability factors.

Economic

Economic constraints include time and cost.

The deadline by which the user or client needs to have the solution operational will define the time available to design and develop the solution. The more time available, the more time there is to complete an in-depth analysis and detailed designs, and to develop advanced features of the solution. The shorter the timeframe, the faster each stage in the problem-solving methodology needs to be completed.

Meanwhile, the funds available to complete the project may affect the hardware and software (digital systems) available for use, the number and range of staff who are available to work on the solution and even the data used as input, if the required data sets need to be purchased.

Both a lack of time and a lack of money may result in a re-evaluation of the user's requirements, or a re-evaluation of how those requirements can be achieved.

Technical

Technical constraints are constraints related to the hardware and software available for the project. Available hardware and software, memory and storage capacity, processing and transmission speeds and security concerns are all examples of possible technical constraints.

For example, developers need to keep in mind that smartphone users may not always have access to a high-speed network connection, so they need to ensure that any animated data visualisation solution does not require a large amount of bandwidth to download and view.

Social, legal and usability

Non-technical constraints relate to areas other than hardware and software. Usability and the user's level of expertise (social) are examples of non-technical constraints. For example, if a solution is being developed for users with little digital systems expertise, this may restrict the inclusion of requirements that would involve complex manoeuvres. Creating a solution for a child audience may also restrict the method used to input data into the solution.

Legal requirements are another type of non-technical constraint. Privacy laws may restrict features linked to displaying personal data in the solution, or to collecting data from the device of someone using your solution. Copyright laws may restrict features that allow other users to upload content to the solution.

Scope

The **scope** outlines the boundaries or parameters of the solution, so that all stakeholders are aware of exactly what the solution will contain. The scope of the solution consists of two elements: what the solution will do and what the solution will not do.

1.6 THINK ABOUT SOFTWARE DEVELOPMENT

List three other technical constraints developers of smartphone apps need to consider when developing a product.

SCHOOL-ASSESSED TASK TRACKER

☐ Project plan	☐ Justification	☐ Analysis	☐ Folio of alternative designs ideas	☐ Usability tests	☐ Evaluation and assessment	☐ Final submission

9780170440943

Many house and land package contracts state exactly what is included in the package as well as what is not included in the package. For example, tiles on the floor of the kitchen are included, but the garden will not be landscaped.

THINK ABOUT SOFTWARE DEVELOPMENT 1.7

What problems do you think a clear scope of solution can avoid later in the project?

What the solution will do

What the solution will do is a list of all the solution requirements (both functional and non-functional) that will be included in the solution.

What the solution will not do

What the solution will not do is a list of all the solution requirements that will not be included in the solution.

Usually these are solution requirements that were initially sought by the client, but that, because of constraints, have been left out of the solution project.

At the start of the project, outlining what will and will not be included in the solution can help prevent arguments between the client and the developer later in the project.

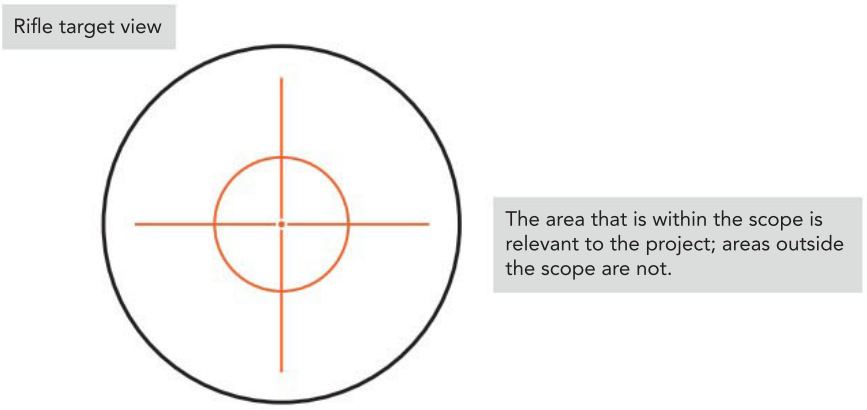


FIGURE 1.14 Scope of solution

An example of a scope of solution would be: The solution will display population data of towns in Victoria in a visual format and graphically represent the distances between those towns. It will be created to be user-friendly, and for privacy reasons, it will not display any personal details.

The requirement that allows the user to zoom in on a particular region or town will not be included in the project because of economic factors but may be added at a later stage.

In a design brief, aspects of the solution that are within scope are described without technical detail, and may consist of general descriptions of what the solution will contain in relation to functional and non-functional requirements. Determining what may be out of scope normally occurs during the analysis stage, such as deciding what functionality may be delayed for later releases due to time constraints.

Representing designs

Once a software requirements specification has been completed, it is important that considerable time is spent designing the software that is going to be written. This helps reduce the time and effort that goes into writing the software, as problems are normally resolved before any code has been written. There is nothing worse than needing to rewrite code due to an issue that could have been resolved in the design stage!

Some common methods of representing designs are to use data dictionaries, object descriptions, mock-ups and pseudocode. Each of these methods has a different purpose in the design stage.

9780170440943

Data dictionaries

A **data dictionary** is used to plan the storage of software elements including variables, data structures, and objects such as GUI textboxes or radio buttons. A data dictionary should list every variable's name and data type or structure. It may also include the data's purpose, source, size, description, formatting and validation.

TABLE 1.8 Data dictionary

Name	*Type*	*Format*	*Purpose*
customerId	integer	999999	Unique identifier for a customer
postCode	string	9999	Postcode for a suburb
userActive	boolean	True/False	Stores if user is active or not
totalOrderCost	floating point	99.99	Total cost of an order

Data dictionaries are valuable when code needs to be modified later by other programmers and the purpose of a variable is unclear. You will learn more about data dictionaries in chapter 4.

Object descriptions

An **object description** is a way of describing all of the relevant properties, methods and events of an object.

```
OBJECT: txtName
PROPERTIES
Class: textbox
Left position: 300
Width: 500
Font: Arial
Justification: left
Visible: yes
Font colour: black
METHODS
Cut: save cut text to disk
EVENTS
Keypress: if key is CTRL+[ set text justification to left.
```

FIGURE 1.15 Example of an object description

Object descriptions are valuable when code needs to be modified later by other programmers and the properties of the object are unclear or unknown.

Mock-ups

If software will be used directly by people (rather than running hidden deep in the OS), it needs an **interface** – a place where people can control the program, enter data and receive output. A successful interface must be carefully designed to ensure it is usable and clear.

SCHOOL-ASSESSED TASK TRACKER

☐ Project plan	☐ Justification	☐ Analysis	☐ Folio of alternative designs ideas	☐ Usability tests	☐ Evaluation and assessment	☐ Final submission

9780170440943

To design an interface, use a **mock-up**, which is a sketch showing how a screen or printout will look. A mock-up should typically include the following features:

- the position and sizes of controls such as buttons and scroll bars
- the positions, sizes, colours and styles of text such as headings and labels
- menus, status bars and scroll bars
- borders, frames, lines, shapes, images, decoration and colour schemes
- vertical and horizontal object alignments
- the contents of headers and footers.

In VCE Software Development you are not required to use software to create your mock-ups. You may use software if you wish, but you can also create them by hand using pen and paper.

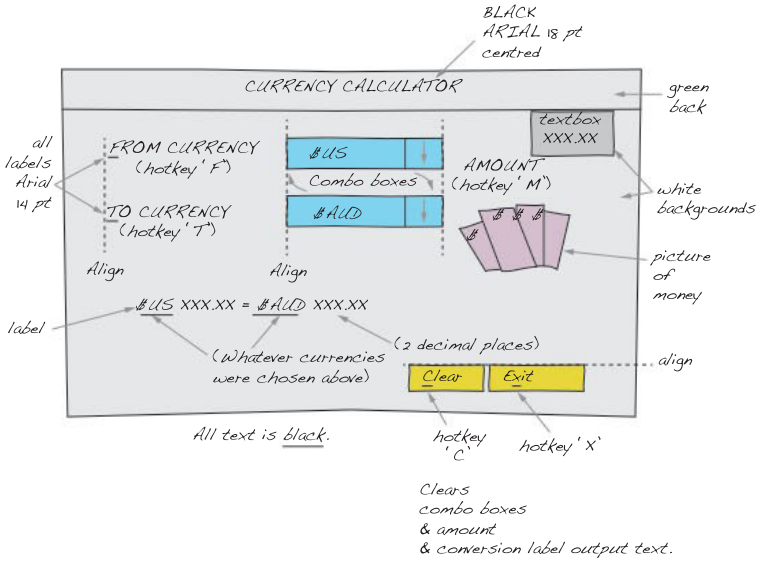


FIGURE 1.16 A mock-up of a screen interface

A mock-up can be considered successful if you can give it to another person and they can create the interface without needing to ask you questions.

Pseudocode

Writing an algorithm in source code is slow. An algorithm written in source code also limits itself to use in only one compiler. **Pseudocode**, also known as Structured English, is a quick, flexible and language-independent way of describing a calculation strategy – halfway between English and source code. Once the algorithm is sketched out in pseudocode, it can be converted into source code for any desired programming language.

A good algorithm can be extremely valuable. A clever strategy can make software run twice as quickly or use half the amount of RAM. An ingenious idea can lead to the development of a program that was previously considered impossible. For example, Google's PageRank completely changed the way the world searched the internet, and made its inventor billions of dollars. The invention of public key encryption finally cracked the age-old problem of how to encode and transmit secrets without having to also send an unlocking key, which could be intercepted.

The following pseudocode determines if a year is a leap year.

```
IF (year is divisible by 4 AND NOT divisible by 100)
        OR (year is divisible by 4 AND 100 AND 400) then
    it's a leap year
ELSE
    it's not a leap year
ENDIF
```

FIGURE 1.17 Pseudocode to determine leap years

9780170440943

The rules of pseudocode

What are the rules of pseudocode? Easy: there are none. As long as the intention of the calculation is clear, it is good pseudocode. If not, it is bad.

It is important to ensure that you specify assignment (the storage of a value) using the ← symbol rather than the equals sign (=) that is used in algebra and in most real programming languages. For example:

`isLeapYear ← True`

The equals sign is reserved for logical comparisons, such as:

IF `B=0` **THEN** `SoundAlarm()`

Common features found in pseudocode include:

- iterations/loops, such as **WHILE/ENDWHILE** and **FOR/NEXT**
- condition control structures, especially **IF/ELSE/ENDIF** blocks
- logical operators – **AND, OR, NOT, TRUE** and **FALSE**
- arrays, such as Expenses[3]
- associative arrays, such as Expenses["Gary"]
- records and fields, such as Customer.firstName, where Customer is the record and firstName is the field
- arithmetic operators (+ – * /) and the familiar order of operations, as used in Year 7 mathematics and Microsoft Excel spreadsheet formulas.

Pseudocode punctuation and the names of key words are largely up to you, so long as it is clear what you mean. For example, it does not really matter if you prefer **WHILE/WEND** or **WHILE/ENDWHILE**.

To 'Get data from keyboard', you could use **INPUT**, **GET**, **FETCH**, or another keyword. To read data from a disk file, you could choose **INPUT**, **GET**, **READ** or something else. To avoid ambiguity, you could explain your pseudocode's conventions using comments within the pseudocode. Comments can be prefixed with a hash, #, or included in parentheses/curly brackets, {}.

```
# GET reads the keyboard.
# READ loads data from a disk file.
# DISPLAY shows output on screen.
# WRITE saves output to a file.
DISPLAY "What is your name?"
GET UserName
OPEN FILE "Users.txt"
READ data for UserName
IF new data exists THEN
    WRITE new data to file
ENDIF
```

FIGURE 1.18 Pseudocode to add new users to a file

Files

Computer files are resources that allow data to be recorded on any type of storage device in a computing system. Files are therefore critical for the operation of almost every software solution if the program needs to save information to be retrieved later.

TABLE 1.9 File operations

open	file.open (filename, mode)
close	file.close ()
read	file.read ()
write	file.write (character)

SCHOOL-ASSESSED TASK TRACKER

☐ Project plan	☐ Justification	☐ Analysis	☐ Folio of alternative designs ideas	☐ Usability tests	☐ Evaluation and assessment	☐ Final submission

9780170440943

There are two types of files: **text files** and **binary files**. Text files store data as easily readable plain text, while binary files store data in binary form, such as with images and sound. Binary files are not easily readable and are therefore more secure than text files.

Files can be opened using different modes, such as read, write, append and binary.

Plain text files

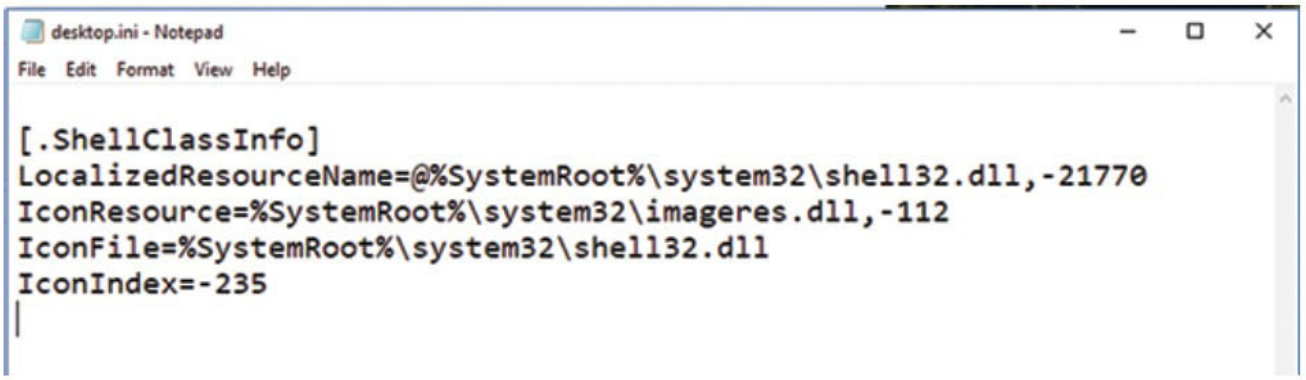
A **plain text file** is a structured file that contains characters of readable data. This data can be structured with spacing, new lines and tabs, but can only be read as character and string data types.

Plain text files are commonly used for configuration settings or for storing small amounts of data in simple software programs.

While plain text files that are stored in a computer system can be opened and read by a human, they are not typically designed for human readability. Instead, they are designed for fast processing and reading by computer programs. This means that a plain text file often lacks comments, headings and sub-headings that would make it more coherent for a human.

```
Paulo    1.23
Shehara  1.35
Phoebe   1.21
Tuan     1.61
```

FIGURE 1.19 An example plain text file of names and heights



©Used with permission from Microsoft

FIGURE 1.20 An example plain text configuration file from a Windows 10 system

Delimited files

A particular type of text file is a delimiter-separated value (DSV) text file, which is a text file where data values are separated by a programmer-selected character. This character is referred to as the **delimiter**. The most common delimiters used in delimited files are commas, tabs and colons. **Delimited files** allow for the storage of two-dimensional arrays in a structured, readable format. When a comma is used as a delimiter in a delimited file, the file is referred to as a *comma-separated value* file, or **CSV** file.

```
PALADINO,Nathan,11,M,PAL0011,11A,MA071 G,IT011 B,AC011 A,PE011 A,EN011 G,BM011 B
CARNUCCIO,Lorenzo,11,M,CAR0022,11A,AC011 B,EN011 D,ME011 A,IT011 A,MA111 E,MA071 B
BRETHERTON,Jessica,11,M,BRE0033,11C,EN011 F,ME011 B,SA011 A,IT011 A,PY011 D,BM011 B
VEAL,Carena,11,M,VEA0044,11C,IT011 B,AC011 A,EN011 C,HI031 A,LS011 B,BM011 B
KHA,Ric,11,M,KHA0055,11B,MA071 G,IT011 B,DT011 B,PH011 A,EN011 G,MA111 A
```

FIGURE 1.21 An example CSV file containing student and subject data

In programming, delimited files are very useful when storing small amounts of data. When there is a lot of data, however, loading, reading and writing to a delimited file is inefficient, as it is very slow. Delimited files are also not secure – anyone opening the file can read its contents. For this reason, they are not suitable for storing sensitive data, such as usernames and passwords, financial details or medical details. This risk to security can be reduced if sensitive data is encrypted.

XML files

An eXtensible Markup Language (**XML**) file is one that has been created using a set of rules for encoding the file into a format that can be read by both a human and a computer program. XML makes it easier to store and transport data within a system and between systems, as it is based on a set of standards and conforms to published conventions. XML was designed to be as self-descriptive as possible, which increases human readability.

Comprehensive documentation outlining XML

XML files contain a **prolog**, which is information that appears before the start of any data in the XML file. It includes information that applies to the XML file as a whole, such as the version of XML it uses and the character encoding of the data within it.

XML is very similar to HTML, but a key difference is that XML has no pre-defined tags. Instead, XML tags are determined by the person who creates the XML file. There are, however, types of tags, referred to as elements, that are meaningful within the XML file.

An XML file contains an XML **tree**, which is the set of elements contained within the file. The tree begins with a **root element** that is a **parent** to **child elements**. These child elements are sub-elements of the root, but any element can contain sub-elements. This makes the structure of an XML file hierarchical, using the analogy of a family tree.

An XML element can contain attributes, text or any other element.

TABLE 1.10 XML element types and characteristics

Root	The first element in an XML tree, and parent to all other elements. There can only be one root element. In Figure 1.22, this is represented by the <recipes> element.
Parent	Any element that contains sub-elements. An example in Figure 1.22 is the <recipe> element.
Child	Any sub-element to another element. An example in Figure 1.22 is the <ingredient> element.
Sibling	Any sub-element on the same level as another sub-element is a *sibling* to it. The <time> and <serve> elements in Figure 1.22 are siblings to each other.
Attribute	An element can contain one or more attributes. Attributes must be enclosed in quotation marks. An example of an attribute in Figure 1.22 is the 'name' attribute inside the <recipe> element.
Text	An element can contain text content. An example of text content in Figure 1.22 is '40 minutes' inside the <time> element of the recipe for 'Fudge Choc-Cherry Biscuits'.

While elements and attributes are user-defined, some naming rules still apply. Elements are **case-sensitive**, must start with a letter or an underscore, cannot start with the letters 'xml' and cannot contain spaces. They can contain letters, numbers, hyphens, underscores and full stops.

SCHOOL-ASSESSED TASK TRACKER

☐ Project plan | ☐ Justification | ☐ Analysis | ☐ Folio of alternative designs ideas | ☐ Usability tests | ☐ Evaluation and assessment | ☐ Final submission

9780170440943

```
<?xml version="1.0" encoding="ISO-8859-1"?>
- <recipes>
  - <recipe name="Chocolate Cake">
        <ingredient>185g dark eating chocolate, chopped coarsely</ingredient>
        <ingredient>1/3 cup (35g) cocoa powder</ingredient>
        <ingredient>1 2/3 cups (410ml) boiling water)</ingredient>
        <ingredient>250g unsalted butter, softened</ingredient>
        <ingredient>2 cups (440g) firmly packed dark brown sugar</ingredient>
        <ingredient>4 eggs</ingredient>
        <ingredient>1 tsp vanilla extract</ingredient>
        <ingredient>3/4 cup (180g) sour cream</ingredient>
        <ingredient>1 cup (150g) plain (all purpose)  flour</ingredient>
        <ingredient>1 cup (150g) self raising flour</ingredient>
        <ingredient>1 tsp bicarb (baking soda)</ingredient>
        <time>1 hr 45 min and refrigeration</time>
        <serve>10</serve>
    </recipe>
  - <recipe name="Anzac Biscuits">
        <ingredient>125g butter, chopped</ingredient>
        <ingredient>2 tbsp golden syrup</ingredient>
        <ingredient>1/2 tsp bicarb soda</ingredient>
        <ingredient>1 cup (90g) rolled oats</ingredient>
        <ingredient>1 cup (150g) plain flour</ingredient>
        <ingredient>1 cup (220g) caster sugar</ingredient>
        <ingredient>3/4 cup (65g) coconut</ingredient>
        <time>Not specified</time>
        <serve>45</serve>
    </recipe>
  - <recipe name="Brandy Snaps">
        <ingredient>1/4 cup (60ml) golden syrup</ingredient>
        <ingredient>90g butter, chopped</ingredient>
        <ingredient>1/3 cup (50g) plain flour</ingredient>
        <ingredient>1 tsp ground ginger</ingredient>
        <time>Not specified</time>
        <serve>20</serve>
    </recipe>
  - <recipe name="Honey and Coconut Muesli Slice">
        <ingredient>2 1/2 cups (225g) rolled oats</ingredient>
        <ingredient>1 cup (35g) rice bubbles</ingredient>
        <ingredient>1/2 cup (40g) shredded coconut</ingredient>
        <ingredient>1/2 cup (70g) slivered almonds</ingredient>
        <ingredient>1 tbsp honey</ingredient>
        <ingredient>395g (14 ounces) canned sweetened condensed milk</ingredient>
        <time>50 minutes</time>
        <serve>36</serve>
    </recipe>
  - <recipe name="Fudge Choc-Cherry Biscuits">
        <ingredient>250g butter, softened</ingredient>
        <ingredient>1 tsp vanilla extract</ingredient>
        <ingredient>3/4 cup (165g) caster sugar</ingredient>
        <ingredient>3/4 cup (165g) firmly packed brown sugar</ingredient>
        <ingredient>1 egg</ingredient>
        <ingredient>2 cups (300g) plain flour</ingredient>
        <ingredient>1/4 cup (25g) cocoa powder</ingredient>
        <ingredient>1 tsp bicarb soda</ingredient>
        <ingredient>1/2 cup (25g) shredded coconut</ingredient>
        <ingredient>1/4 cup (50g) glace cherries, chopped coarsely</ingredient>
        <ingredient>200g dark eating chocolate, chopped coarsely</ingredient>
        <ingredient>200g milk eating chocolate, chopped coarsely</ingredient>
        <time>40 minutes</time>
        <serve>40</serve>
    </recipe>
</recipes>
```

FIGURE 1.22 An XML file containing ingredients for recipes

9780170440943

The advantages of using an XML file over a plain text file are that XML is industry standard, widely used and cross-platform. It allows rules to be set and used on data in a way that text files cannot. XML also allows storage of data that does not rely on a user interface – the same data can be displayed in different formats and interfaces.

XML files are used for many different purposes, including:

- storing data – for internal and/or external systems
- storing configuration information
- storing user interface details
- moving and sharing data between internal and/or external systems.

These purposes are particularly useful for cross-platform and cross-system applications. Using an XML file ensures that data received from a source is in an 'as expected' format. This ensures data integrity across systems.

Programming languages

Programming languages are used to give instructions to computer processors so they can calculate useful information or carry out tasks for humans. Whether your phone is streaming music, your car is turning on its anti-skid braking, or McDonald's is calculating staff wages, programming languages are needed.

Like human languages, there are many programming languages, each with distinctive grammar, punctuation and vocabulary. Most programming languages have special abilities or strengths that make them more useful than other languages for a particular task.

Professional programmers know a handful of languages and choose the best language for each job, based on its strengths and weaknesses. Choosing which languages to learn is a big decision, but remember that learning one language makes it easier to learn others. The most popular programming languages include C (C++ or C#), Python, Java, JavaScript, Perl and PHP, SQL, Visual Basic and Swift.

- C, C++ or C# are used for writing low-level systems and utilities and fast applications. This could include operating systems, embedded microcontroller programs, web-based applications and games.
- Python is an interpreted object-oriented programming language used for web and app development.
- Java is used for web applications and web services and for building Android apps.
- JavaScript is a client-side scripting language for websites.
- Perl and PHP are used in website and network programming.
- SQL, or structured query language, is a scripting language for database programming.
- Visual Basic and Visual Basic .NET are used widely to create applications for Windows-based computers.
- Swift is a programming language for macOS, iOS, watchOS and tvOS.

While programming languages may differ, they all do basically the same job. They control a digital system such as a computer, tablet or smartphone.

Programming languages differ in the amount of direct control they give over a computer's hardware and operating system. With a high-level language such as Visual Basic or Python,

SCHOOL-ASSESSED TASK TRACKER

☐ Project plan	☐ Justification	☐ Analysis	☐ Folio of alternative designs ideas	☐ Usability tests	☐ Evaluation and assessment	☐ Final submission

9780170440943

programmers avoid having to worry about complex details of the structure of actual disk files or where data is stored in memory. High-level languages are simpler to use, but lack the control of complex but more difficult to learn low-level languages. Conversely, a low-level language such as assembly or machine code requires more skill and knowledge from the programmer, but allows more direct control of the workings of a computer.

High- and low-level programming languages each have their uses. To write a simple alarm clock program, a high-level language is fine. To write firmware for a micro-controller, a low-level language is more appropriate.

Internal documentation

One of the most important yet easily forgotten aspects of writing good code is including meaningful **internal documentation**. Internal documentation is the notes and comments written by a programmer within the code itself. It includes information about the program as a whole, as well as about each of the classes, functions, methods, objects, algorithms, etc. within it. It is often combined with meaningful, well-named variables to create manageable and effective code.

Internal documentation has no impact on the compilation or running of the code itself. It exists only to provide context and important information about the code. Often it is most useful when a programmer is reading through code that they did not write themselves, or that was written a long time ago. This means that the programmer does not need to rely on memory alone or on interpreting complex algorithms to understand how the program works. Well-written internal documentation saves time, as it reduces effort on the part of the developer, making it a core feature of efficiency in creating and maintaining software solutions.

Internal documentation conventions

While there is no single set of conventions for internal documentation, there are many common elements of internal documentation that should be included. These are:

- a **header comment** , containing the name of the file, a brief description of the program, the author's name, and the date the program was first created
- documentation of classes and methods, describing their behaviour and how they are used, including any expected inputs and outputs and their respective data types
- function and subroutine comments, describing their purpose, as well as describing all inputs and outputs and their respective data types
- single-line comments, providing brief summaries of portions of code
- multi-line comments, explaining a complex algorithm within the code itself
- descriptions of how to test aspects of the software
- extra information on upgrades, changes or enhancements made to the program.

While internal documentation can include comments related to revisions and new versions, this does not replace the need to use an effective **version control system**.

When writing internal documentation, it is important that the comments made within the source code are well formatted so they can be easily read. Comments should be separated from code by a blank line before the comment. For multi-line comments, a blank line should be included before and after the comment. Comments should be vertically aligned with the

Version control is the management of changes to source code files and other project-related documents throughout the duration of a software project. Version control systems typically run as stand-alone programs or web-based systems that not only help track changes to documents, but also allow for more than one developer to work on source code at the same time. They also enable a developer to revert back to previous versions of source code.

Git/GitHub version control

indent level of the current code to make it clear which code it is associated with. Depending on the programming language used, comments must be enclosed using special characters. Table 1.11 includes a list of the characters used in some popular programming languages.

TABLE 1.11 Common programming language commenting conventions

C, C++, Java, JavaScript, Swift	// this is a single line comment /* This is a multiple line comment */
Python	# this is a single line comment
PHP	# this is a single line comment // this is also a single line comment /* This is a multiple line comment */
XML, HTML	<!-- this is a single line comment --> <!-- This is a multiple line comment -->
VB, VB.Net	‘ this is a single line comment

Many programming languages have tools that make creating internal documentation easier. There are also tools that extract internal documentation to create reference manuals and online documentation for users.

Doxygen

While it is important to include internal documentation in all software modules, the comments that are included should be meaningful and non-trivial. Unless it is being used for teaching purposes, internal documentation should not simply state or re-state what is occurring in the code, particularly if this is clearly apparent in the code itself.

For example, the following commented Python code would be considered trivial, as the code comments do little more than re-state what the next line of code does.

```
def readFromCSV( csvFileName ):

    # open the CSV file for reading
    csvFile = open(csvFileName, "r")

    # read all of the lines in the csv file
    csvContents = csvFile.readlines()
    return csvContents
```

FIGURE 1.23 Poorly commented code

An example of better comments is shown in Figure 1.24. Notice that the code comments provide information about the function, its inputs, and its outputs, including reference to data types.

9780170440943

```
# Function: Reads a CSV file and returns its contents
# Input: String, filename
# Output: Array of Strings, contents of the file
def readFromCSV( csvFileName ):

    csvFile = open(csvFileName, "r")
    csvContents = csvFile.readlines()
    return csvContents
```

FIGURE 1.24 Appropriately commented code

Naming conventions

A **naming convention** is a set of rules that is used when creating variables, subroutines, functions, methods, objects, classes, etc. in programming source code, as well as in internal documentation. It is a consistent and meaningful way of labelling each of these elements so that they are easily read and understood. The most useful naming conventions tell a programmer the purpose of an element and, if relevant for the programming language, its data type or structure.

Each programming language tends to have a set of language-specific conventions to follow when naming elements. For example, Microsoft.NET, Python and Swift all use a convention called 'camel case' within its code. Two other common naming conventions are snake case and Hungarian notation.

Camel case

Camel case, also known as camel caps or lowerCamelCase, uses compound words and phrases as a naming convention, where each word after the first begins with a capital letter. For example, camelCase. No spaces or punctuation are included when naming variables and other elements. While multi-word variable and function names are useful, it is important that these are kept as short as possible, while remaining meaningful. Often, this is achieved through abbreviating some of the words in the compound phrase. A variable named `firstNameOfEmployeeWhoIsPartTime` is not as effective as one named `firstNameEmplPT`, as it is too long. Writing and reading code using the longer version of this variable would be very tedious. Camel case is one of the most common naming conventions used in modern programming, in particular when programming using a dynamically typed language.

Snake case

Snake case is very similar to camel case, but instead of compounding phrases into a single word without spaces, it joins each word in the phrase using an underscore. For example, `snake_case`. Many programmers prefer snake case over camel case because the underscores separating each word make it easier to read the variable, method and function names.

9780170440943

Hungarian notation

Hungarian notation, in particular a variety referred to as Systems Hungarian, is similar in style to camel case, in that it compounds words and phrases, without spaces, and each word after the first begins with a capital letter. Hungarian notation also adds a *prefix*, that is, an initial letter sequence, before the name of the variable. This letter sequence represents the data type or structure of the variable. For example, `iNumEmployees` to represent the number of employees as a whole integer. This can be useful when programming using programming languages that are not dynamically typed, as it immediately tells a programmer what data type or structure they are handling when they read the variable name.

TABLE 1.12 Some Hungarian notation prefixes

iNumEmployees	Integer
arrEmployees	Array
strEmployee	String
chEmployeeGender	Character
fpEmployeeHeight	Floating point
fnEmployeeFunction	Function

Hungarian notation is a flexible naming convention, with many programmers creating user-defined prefixes that are meaningful for the programming language they are using. For example, while classes and objects were not considered in the initial Hungarian notation prefixes, many programmers use 'cl' as the prefix to represent a class and 'obj' for an object. With the increasing prevalence of dynamically typed programming languages, however, most programmers prefer to use the camel case or snake case naming convention over a modified Hungarian notation.

9780170440943

1 CHAPTER SUMMARY

Essential terms

32-bit computer system a computer system with 32 bits of memory addresses

64-bit computer system a computer system with 64 bits of memory addresses

array a list of elements indexed by position. In most programming languages the first element has index zero.

associative array similar to an array; information is stored in key-value pairs

binary file a computer-readable file, such as executable programs, images and sound

boolean a data type with one of two possible values, 0 and 1, usually referred to as False and True, respectively

camel case a naming convention in programming where each word or abbreviation after the first in a phrase begins with a capital letter; there are no spaces or punctuation

case-sensitive a program's ability to distinguish between upper-case and lower-case letters

character a data type representing any single meaningful unit, such as a letter, a number, a punctuation mark, a symbol, or even a space

character encoding a code that allows a computer program to interpret binary digits (0s and 1s) into meaningful units representing characters and numbers. For example, ASCII, UTF and Unicode are types of character encoding.

child element any sub-element of a parent element in an XML file

class a program code template for creating objects in object-oriented programming languages

collision when two different input values to a hashing function output the same hash value

constraint a restriction on what can and cannot occur in the creation of a software solution, external to the solution itself

CSV a comma-separated value file, which is a delimited file, separated by commas

data dictionary a set of information that describes elements within software, such as variables, data structures and objects

data structure a method of organising data to allow particular operations to be performed on them efficiently

data type a method of classifying a variable to determine the data that variable can contain, as well as how that variable can be manipulated

delimited file a text file where data values are separated by a programmer-selected character

delimiter the character used to separate data values in a delimited file

dequeue removing and returning a value from the start of a queue

dictionary an associative array, also referred to as a map or symbol table

enqueue inserting a value at the end of a queue

field a single data item in a record, e.g. FamilyName

first in first out (FIFO) the first element in a queue is the first element out of the queue

floating point computer representation of real numbers, with decimal places

hard-coding to include fixed data in a program that cannot be changed during run-time and can only be changed by modifying the program source code

hash function a function that takes a key value and returns another, related, value that is normally smaller than the original value

9780170440943

hash table a data structure that uses a hash function to map keys to values by computing an index that is related to, but smaller than, the initial key

header comment a set of meaningful comments at the top of a source code file, outlining information such as the name of the file, its purpose, the author's name and the date of creation

Hungarian notation a naming convention in computer programming where the name of the variable or function determines its purpose and its data type or structure

imperfect hash a hash function where two or more keys can be computed to have the same hash index

inheritance a method of basing an object or class on another object or class, taking on its attributes and methods and potentially extending upon them

instantiation in object oriented programming, the process by which an object is created from a class

integer a data type representing whole positive and negative numbers

integer overflow a condition occurring as a result of a mathematical operation where the output exceeds the maximum or minimum integer value that can be stored on that computer system

interface within software, the place where people control the program, enter data and receive output

internal documentation notes and code comments contained within source code that describe the code

last in first out (LIFO) the last element in a stack is the first element out of the stack

linked list a data structure containing an ordered set of elements in which each element is connected to the next element in the list

mock-up a sketch showing how a screen or printout will look, used to aid in the design of an interface

naming convention an agreed set of rules by which to name source code elements such as variables, functions, classes, methods and objects

node a basic unit of a data structure that may contain data and/or link to other nodes

non-technical constraints limitations relating to areas other than hardware and software: social, legal and usability

numeric a data type consisting of whole numbers, referred to as integers, and decimal numbers, referred to as floating points

object any instantiated class that a program can inspect and/or change, in terms of appearance, behaviour or data

object description a way of describing all of the relevant properties, methods and events of an object

parent element any element in an XML file that contains at least one sub-element

perfect hash a hash function where no two keys can be computed to have the same hash index

plain text file a structured file that contains characters of readable data

pointer a programming language element that stores the memory address of another data value located in memory; the pointer 'points' to that memory space

pop an element of data removed from the top of a stack, moving all remaining stack elements up one place

prolog the information in an XML file that appears before the start of the document's contents, including information such as the XML version and character encoding that is being used

pseudocode code that designs algorithms in a clear, human-readable, language-independent format

push an element of data inserted at the top of a stack, moving all current stack elements down one place

queue a 'first in first out' data structure storing elements to be processed in order

RAM random access memory; a type of computer memory that can be accessed randomly; it is most often volatile memory that is lost if power is removed

record a complete set of fields relating to an entity, such as a person

root element a parent element to all other elements in an XML file

scope the boundaries or parameters of the solution – what it will do and what it will not do

snake case a naming convention in programming where each word or abbreviation in the middle of a phrase is joined using an underscore

solution requirements what the client needs from the solution in relation to its features

9780170440943

1 CHAPTER SUMMARY

stack a 'last in first out' data structure

string a data type representing a set or sequence of character data types

struct record used in database systems and programming languages

technical constraints constraints related to the hardware and software available for the project

text file a structured file containing sequences of characters that are not encrypted, such as a plain text file or CSV file

tree the structure of an XML file that contains a root element and all of its sub-elements

variable a method of storing and labelling data to be referenced and manipulated in a computer program

version control system a software product that manages the revisions, changes and parallel editing of source code and its related documentation

XML eXtensible Markup Language, a metalanguage that allows for user-defined tags and rules for encoding documents in a format that is readable by humans and machines

Important facts

1. **Data types** are consistent across all programming languages.
2. **Variables** can be classified as particular data types and structures.
3. **Integers** are positive and negative whole numbers; **floating point numbers** can have decimal places.
4. It is important to know the computer architecture on which a program will run before designing and developing a software solution, as some data types differ depending on whether the computer runs on a 32-bit or 64-bit system.
5. A set or sequence of characters is also known as a **string data type**.
6. **Boolean values** are 0 and 1, but sometimes coded as true or false in a programming language.
7. **Data structures** are more complex than data types.
8. **Arrays** start at index value 0 in almost all programming languages.
9. Arrays traditionally contain elements of the same data type, but this depends on the programming language selected.
10. **Dictionaries** and **hash tables** are types of associative arrays.
11. **Queues** are first in first out (FIFO), **stacks** are last in first out (LIFO).
12. **Records** are collections of related data (fields) that may or may not have the same data types.
13. **Classes** are blueprints, and **objects** are instantiations of those blueprints.
14. Classes can be extended upon using **inheritance**.
15. **Objects** contain methods, a function or subroutine, and events, called when an object's state changes.
16. A **design brief** is typically written during the analysis stage of the problem-solving methodology.
17. **Solution requirements** describe what a client needs from a solution; they are general rather than technical descriptions.
18. **Solution constraints** limit or restrict solution requirements.

9780170440943

19 The **scope of a solution** is the boundary or parameters of the solution that outlines what a program will do and what it will not do.

20 **Designs** can be represented using tools such as data dictionaries, object descriptions, mock-ups and pseudocode.

21 **Data dictionaries** are valuable as references when modifying code.

22 **Object descriptions** describe all of the relevant properties, methods and events in an object.

23 **Mock-ups** are annotated visual representations of the user interfaces of software solutions.

24 **Pseudocode** is a way of representing algorithms using structured English that does not rely on the syntax of any programming language.

25 **Text files** are easily readable by a human; **binary files** are not.

26 **Plain text files** are typically structured using spacing, new lines or tabs.

27 **Delimited files** are a way to store two-dimensional arrays in a structured, readable format.

28 **XML** is a powerful markup language that allows for easy transportation of data between systems.

29 **Programming languages** give instructions to computer processors so they can carry out tasks for humans.

30 While programming languages may differ, they all do basically the same job: they control a digital system such as a computer, tablet or smartphone.

31 **Internal documentation** should be relevant, consistent and non-trivial.

32 **Naming conventions** make source code easier to read, increasing its effectiveness.

9780170440943

TEST YOUR KNOWLEDGE

Review quiz

Data types and structures

1 Select the most appropriate data types and structures for the following data:

a `222`

b `2.95`

c `True`

d `panda`

e `019234`

f `Customer: { Phoebe, Corp, 08/08/2018, 123 Fourth Street, Fifthsville, VIC, 3888 }`

g `Players: { Wanda, Greg, Tuan, Rishad, Dillon, Nicole, Shveta, Ramesh }`

h `Stock: { "potatoes": 300kg, "cauliflower": 344kg, "peas":120kg, "carrots":403kg }`

i `-`

j `0`

2 A computer program runs an algorithm on very large numbers and displays an incorrect output number: 2 147 483 647. Explain what has most likely occurred.

3 What is the difference between a hash table and a dictionary?

4 When would you use a record over an array to store a collection of related values?

5 A cafe would like a new ordering system to process orders so that they are cooked in the order in which they are received. Which data structure is the most appropriate to store kitchen orders?

6 An application to store song playlists has been written so that when a new song is added to the playlist, it is queued so that it plays next, in front of any other song on the playlist. Which data structure is the most appropriate to use for the playlist?

Naming conventions

7 Use the pseudocode below to answer the questions that follow:

```
INPUT strFileName
dictAll ← {}
elTree ← fnReadXmlFromFile(strFileName)
elRoot ← elTree.fnGetRoot()
FOREACH elSubEl IN elRoot
    dictAll[elSubEl.fnGetElement("height")] ←
        elSubEl.fnGetAttrib("height")
ENDFOREACH

RETURN dictAll
```


9780170440943

a Which variable is a string?
b What data structure is dAll?
c What data structure is elRoot?
d What naming convention is being used?

Design briefs

8 What is contained within a design brief?
9 Define 'scope'.
10 Define 'constraints'.
11 Explain how scope can affect the success of a solution.
12 A developer is writing an application for a mobile phone. What are two constraints the solution will have?
13 Why is it important to design a system before writing source code?

Representing designs

14 What is a data dictionary and what is its purpose?
15 What is an object description and what is its purpose?
16 What is pseudocode and what is its purpose?
17 What is the difference between ← and = in pseudocode?
18 State the values that will be returned or displayed in the following examples of pseudocode:

a

```
stkFruit ← { "banana", "cherry",
"mango" }

stkFruit.pop()
stkFruit.push("pear")
stkFruit.pop()
stkFruit.pop()

DISPLAY stkFruit.top()
```

b

```
qFruit ← { "banana", "cherry",
"mango" }

qFruit.dequeue()
qFruit.enqueue("pear")
qFruit.dequeue()
qFruit.dequeue()

DISPLAY qFruit.front()
```

SCHOOL-ASSESSED TASK TRACKER

☐ Project plan	☐ Justification	☐ Analysis	☐ Folio of alternative designs ideas	☐ Usability tests	☐ Evaluation and assessment	☐ Final submission

9780170440943

TEST YOUR KNOWLEDGE

Files

19 What is the difference between a text file and a binary file?

20 What are delimited files?

21 What is a CSV file?

22 What is the difference between XML and HTML?

23 What is the purpose of XML? Provide an example as part of your explanation.

24 Describe a situation where you would use a CSV file over an XML file.

25 Describe a situation where you would use an XML file over a CSV file.

Programming languages

26 Name three high-level programming languages.

27 What is the difference between a high-level programming language and a low-level programming language?

Internal documentation

28 Explain the purpose of internal documentation.

29 What are three conventions of internal documentation that should be included in source code?

30 Does internal documentation slow down a software solution? Explain.

Naming conventions

31 Name two types of naming conventions.

32 Why are naming conventions important in source code?

9780170440943

APPLY YOUR KNOWLEDGE

1 Brainstorm ideas for a problem, opportunity or need for which you are interested in creating a software solution. You may need to ask people in your family or local community for ideas as part of this process. Consider applications relevant to your hobbies and interests as well. Examples of potential projects are:
   - job scheduling system for a small business
   - healthy lifestyle app for a mobile system
   - fitness apps for wearable technology
   - order management system for a retail store
   - automated invoice production
   - competition ladder tournament creation tool.

2 The nature of your problem will determine the contents of your design brief. Begin drafting your design brief by describing the problem, opportunity or need that most appealed to you from question 1. You should check with your teacher to make sure your selection is feasible and appropriate.

3 Complete data collection to determine the constraints of your solution. You will want to consult with a client to gather this information. Ensure you consider technical, economic, social, legal and usability constraints. Document these constraints in your design brief. It may be useful to list and describe the constraints in table format.

4 Complete data collection to determine the scope of your solution. You will want to consult with a client to gather this information. Ensure you consider your constraints when determining scope. Document what is in and out of scope in your design brief. It may be useful to list these as dot points.

5 Complete your design brief. Ensure it includes:
   - **a** details about your client
   - **b** a full description of the problem, opportunity or need
   - **c** constraints relevant to the proposed solution
   - **d** the scope of the proposed solution.

SCHOOL-ASSESSED TASK TRACKER

☐ Project plan	☐ Justification	☐ Analysis	☐ Folio of alternative designs ideas	☐ Usability tests	☐ Evaluation and assessment	☐ Final submission

9780170440943

CHAPTER

2 Development and features of a computer program

KEY KNOWLEDGE

On completion of this chapter, you will be able to demonstrate knowledge of:

Approaches to problem solving

- processing features of a programming language, including classes, control structures, functions, instructions and methods
- algorithms for sorting, including selection sort and quicksort
- algorithms for binary and linear searching
- validation techniques, including existence checking, range checking and type checking
- techniques for checking that modules meet design specifications, including trace tables and construction of test data.

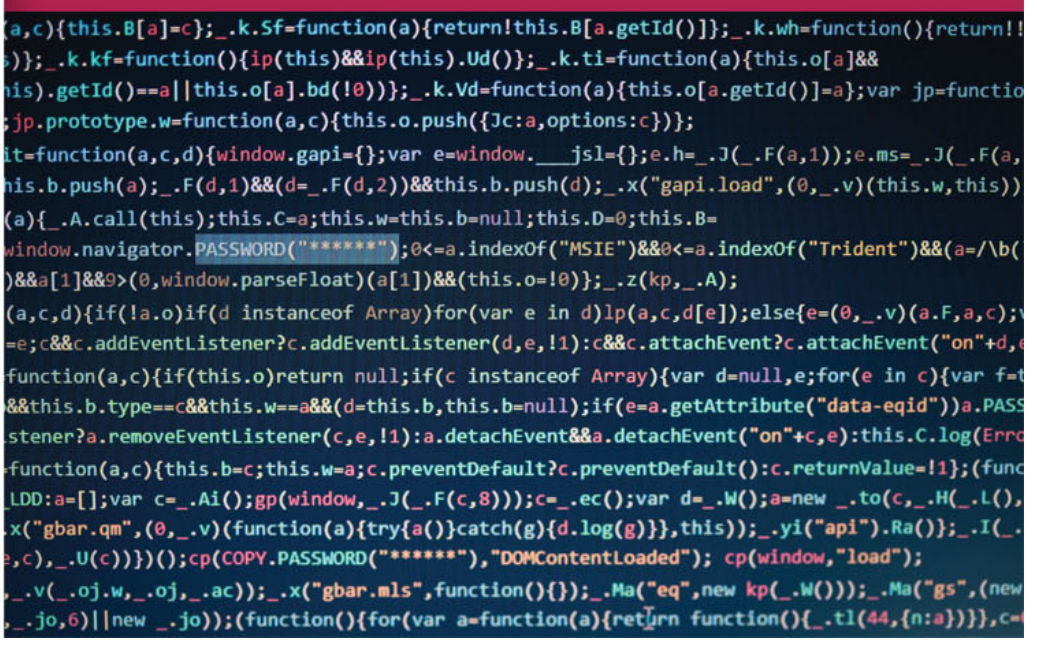
Reproduced from the VCE Applied Computing Study Design (2020–2023) © VCAA; used with permission.

FOR THE STUDENT

The processing features of a programming language are fundamental, as the focus is on the logic of programming, rather than on syntax. Similarly, algorithms written in pseudocode are able to be translated into any language on any platform; this is the benefit of designing an algorithm in pseudocode before writing it in source code. Validation techniques are critical to make sure data that is handled by a computer application is as well-formed as possible. While it is important to take time to develop a software solution, it is equally important to spend time checking that the modules that have been written meet design specifications and are as bug-free as possible. This involves rigorous testing procedures.

FOR THE TEACHER

The focus of this chapter is on the fundamental processing features of a programming language. Students should spend a considerable amount of time putting the theoretical underpinnings of programming logic into practice using their selected programming language, initially writing small pieces of source code that attempt instructions, control structures, methods, functions and classes. Once students are comfortable with these processing features, they should attempt to implement more complex algorithms, such as sorting and searching algorithms. Chapters 1 and 2 form the basis of Unit 3, Outcome 1.



123RF.com/przemekklos

Processing features of a programming language

Variables

Variables are methods of storing data so that they can be retrieved later within a program. Without variables, it would be impossible to reference data once it has been stored in memory. A variable is typically used to store a data type or structure, but it can also be used to store a **pointer** to a function or method. Variables should be named appropriately, following consistent naming conventions.

Instructions

An **instruction** is a unit of code that can be executed by a **compiler** or **interpreter**. There are two types of instructions in programming: definitions and statements.

A **definition** is an instruction that assigns a value to a variable. The first line in Figure 2.1 is an example of a definition.

A **statement** is a single line of code that, when executed, performs a single action. The last line in Figure 2.1 is an example of a statement.

```
a ← 7

PRINT a
```

FIGURE 2.1 Pseudocode example of two types of instructions

Control structures

There are three fundamental control structures in programming: sequences, conditions and iterations.

Sequence

A **sequence** is a set of instructions that executes line by line, a little bit like a recipe. Every line of code in the sequence is run in the order that it is written.

```
ALGORITHM askName()
BEGIN
    PRINT "What is your name?"
    INPUT name
    PRINT "Hello, " + name + ". Nice to meet you."
END
```

FIGURE 2.2 Pseudocode example of a sequence of instructions

In the algorithm shown in Figure 2.2, a sequence of code runs that asks a user for their name, reads the name as input and then greets the user by name. Each line of code in the algorithm is run, in order, only once.

SCHOOL-ASSESSED TASK TRACKER

☐ Project plan	☐ Justification	☐ Analysis	☐ Folio of alternative designs ideas	☐ Usability tests	☐ Evaluation and assessment	☐ Final submission

9780170440943

Conditional

A **conditional statement** is a control structure that allows a programmer to write lines of code that are only run when a particular requirement is met; this is sometimes referred to as a selection control structure. The code within a conditional statement can contain instructions, sequences, other conditional statements or iterations.

Conditions are **boolean**, in that they are run based on the result of a condition being evaluated as either true or false. If the condition evaluates as 'true', the code within that conditional statement is executed. If it evaluates as 'false', it is not executed.

The simplest type of conditional statement is one that tests against a single condition, as seen in Figure 2.3.

```
ALGORITHM printPositive()
BEGIN
    INPUT firstNumber

    IF firstNumber > 0 THEN
       PRINT "The number is positive."
    ENDIF
END
```

FIGURE 2.3 A single condition selection control structure

Alternative execution

Another form of a conditional statement involves an **alternative execution**. This means that if the condition is not met, alternative code will run. For example, in Figure 2.4, a user is asked if they like pie. If they respond with 'yes' they receive a happy comment. If the user does not input 'yes', a sad comment will be printed instead. Note that the user does not need to input 'no' for the sad comment to be printed – any input other than 'yes' will execute the alternative code.

```
ALGORITHM likePie()
BEGIN
    PRINT "Do you like pie?"
    INPUT likePie

    IF likePie = "yes" THEN
            PRINT "Hooray!"
    ELSE
            PRINT "That makes me sad. :("
    ENDIF
END
```

FIGURE 2.4 A condition selection control structure with alternative execution

As you can see in Figure 2.4, the use of 'ELSE' in a conditional statement is optional.

Conditionals with more than one logical expression

A conditional statement is not limited to testing only one logical expression. Theoretically, the number of logical expressions a single conditional statement can test is unlimited. Figure 2.5 contains pseudocode that uses the **logical operator** 'AND' within a single conditional statement to check for two conditions to be simultaneously true.

9780170440943

```
ALGORITHM likePieCake()
BEGIN
    PRINT "Do you like pie?"
    INPUT likePie
    PRINT "Do you like cake?"
    INPUT likeCake

    IF likePie = "yes" AND likeCake = "yes" THEN
            PRINT "Hooray!"
    ELSE
            PRINT "That makes me sad. :("
    ENDIF
END
```

FIGURE 2.5 A simple conditional with more than one logical expression

When writing conditionals with more than one logical expression, it can be useful to construct a **truth table** to make sure that no logic errors have been made. Truth tables use boolean algebra to test each combination of values in a condition. For example, Table 2.1 contains a truth table to check the logical expressions in the pseudocode from Figure 2.5. Because the logical operator connecting the two conditions was 'AND', both conditions need to be true for the conditional statement as a whole to evaluate as true, which is shown when only a single case in the truth table evaluates the whole condition as true.

TABLE 2.1 Truth table to evaluate 'AND'

likePie	likeCake	likePie AND likeCake
True	True	True
True	False	False
False	True	False
False	False	False

If the logical operator connecting the two conditions was 'OR', the resulting truth table can be seen in Table 2.2. In this instance, the use of 'OR' expands the number of cases where the condition would evaluate to true to three, with the only time it evaluates to false being when both conditions are false.

TABLE 2.2 Truth table to evaluate 'OR'

likePie	likeCake	likePie OR likeCake
True	True	True
True	False	True
False	True	True
False	False	False

Truth tables are a systematic method of testing the logic of a conditional statement. They are particularly useful when there are more than two conditions within a single statement.

9780170440943

For example, Table 2.3 contains a truth table for the conditional statement in Figure 2.6. The truth table shows that there are three cases where the entire conditional statement would evaluate to true, and five cases where it would evaluate to false. This would not be easily apparent without constructing the truth table.

```
ALGORITHM multiConditions()
BEGIN
        INPUT a, b, c, d
        IF a < b AND ( b < c OR c < d ) THEN
                PRINT "Condition met."
        ELSE
                PRINT "Condition not met."
        ENDIF
END
```

FIGURE 2.6 A complex conditional with more than one logical expression

TABLE 2.3 Truth table for Figure 2.6

a < b	b < c	c < d	b < c OR c < d	a < b AND (b < c OR c < d)
True	True	True	True	True
True	True	False	True	True
True	False	True	True	True
True	False	False	False	False
False	True	True	True	False
False	True	False	True	False
False	False	True	True	False
False	False	False	False	False

A truth table can be used in conjunction with a trace table to determine if an algorithm is without logical errors. They can also be used to help select test data for testing an algorithm, as discussed later in this chapter.

Chained conditional

The algorithm shown in Figure 2.7 uses a more complex set of conditional control structures (IF/ELSEIF) in order to react to user input when a condition needs to be tested more than once. This is referred to as a **chained conditional**. In this example, the user can select four operations: addition, subtraction, multiplication and division. The program must therefore test the user input four times to see if it matches against the four conditions given.

As the algorithm uses 'ELSEIF', it will check each condition only if the condition prior to it evaluates as false. Without the use of 'ELSEIF', each condition would be run in sequence regardless of whether the condition before it evaluated as true or false. This is an important characteristic of the condition control structure that is often forgotten by programmers, resulting in logical errors in code.

9780170440943

```
ALGORITHM computeOperation()
BEGIN
        PRINT "What is the first number?"
        INPUT firstNumber
        PRINT "What is the second number?"
        INPUT secondNumber
        PRINT "What operation would you like to perform?"
        INPUT operationChosen
        Total ← 0
        IF operationChosen = "add" THEN
                Total ← firstNumber + secondNumber
        ELSEIF operationChosen = "subtract" THEN
                Total ← firstNumber - secondNumber
        ELSEIF operationChosen = "multiply" THEN
                Total ← firstNumber * secondNumber
        ELSEIF operationChosen = "divide" THEN
                Total ← firstNumber / secondNumber
        ELSE
                PRINT "Invalid operation chosen."
        ENDIF

        PRINT "The result is: " + Total
END
```

FIGURE 2.7 Pseudocode example of a chained conditional control structure

```
ALGORITHM ifElseExample()
BEGIN
        PRINT "What is the current temperature?"
        INPUT currentTemp

        IF currentTemp < 10 THEN
                PRINT "It is very cold."
        ENDIF
        IF currentTemp < 20 THEN
                PRINT "It is a little cool."
        ELSE
                PRINT "It is very warm."
        ENDIF
END
```

FIGURE 2.8 Pseudocode example of conditions with logical errors

Consider the pseudocode in Figure 2.8. Assume the current temperature is input as 9 degrees. The first IF condition tests to see if the temperature is less than 10 degrees – as 9 degrees is less than 10 degrees, the algorithm will print 'It is very cold' and then continue in sequence to the next line of code in the program, which is the second IF condition. The second IF condition will check if the temperature is less than 20 degrees: as 9 degrees is less than 20 degrees, it will print 'It is a little cool' and then continue in sequence to the next line of code in the program, which is the ELSE condition. This portion of the code is not run: because the IF statement it is attached to evaluated to true, the ELSE condition has not been met (9 degrees is not greater than 20 degrees).

There is no limit to the number of conditions that can be contained in a chained conditional statement. Chained conditions also do not need to contain an ELSE statement.

9780170440943

It is important to use consistent formatting in your source code. This not only makes the code easier to read, it also reduces the chance of logical errors occurring due to nested conditionals. The pseudocode in Figure 2.9 shows the hierarchy of the conditional statements because of the use of indentation, such as tabs and spacing; it is much easier to read than if no indentations were used.

Nested conditional

Conditional statements can also be placed inside each other. This type of condition is referred to as a **nested conditional**. This control structure is useful when multiple conditions must be handled within the code. For example, Figure 2.9 contains pseudocode where a conditional statement is used to check if it is raining. If it is, another check is made to see if the person has an umbrella. If they do, they use the umbrella, otherwise they get wet. The check for 'hasUmbrella' is nested within the check for 'isRaining'.

```
ALGORITHM checkUmbrellaUsage()
BEGIN
        INPUT isRaining
        INPUT hasUmbrella

        IF isRaining = True THEN
                IF hasUmbrella = True THEN
                        useUmbrella()
                ELSE
                        getWet()
                ENDIF
        ENDIF
END
```

FIGURE 2.9 Pseudocode examples of a nested conditional statement

Switch/Case

A **switch/case** statement, also referred to as a switch statement, is very similar to a chained conditional, in that it allows for multiple conditions to be tested. Figure 2.10 includes an example of the use of switch/case.

```
ALGORITHM computeOperation()
BEGIN
        PRINT "What is the first number?"
        INPUT firstNumber
        PRINT "What is the second number?"
        INPUT secondNumber
        PRINT "What operation would you like to perform?"
        INPUT operationChosen

        Total ← 0
        SWITCH operationChosen
                CASE "add"
                        Total ← firstNumber + secondNumber
                CASE "subtract"
                        Total ← firstNumber - secondNumber
                CASE "multiply"
                        Total ← firstNumber * secondNumber
                CASE "divide"
                        Total ← firstNumber / secondNumber
                DEFAULT
                        PRINT "Invalid operation selected."
        ENDSWITCH

        PRINT "The result is: " + Total
END
```

FIGURE 2.10 Pseudocode example of a switch/case control structure

Not all programming languages have switch/case functionality implemented. In the programming languages that do have it implemented, it is typically more efficient to use switch/case than it is to use chained conditionals.

9780170440943

Iteration

An iteration, also known as a loop, is used to repeat sections of code multiple times until a condition is met. There are four main types of iterations: WHILE loops, DO/WHILE loops, FOR loops and REPEAT/UNTIL loops.

Most programming languages include a method for exiting an iteration early. These are often referred to as 'breaks'.

WHILE loops

A **WHILE loop** is a section of code that is run when, and for as long as, a condition is met. These types of loops are useful when the programmer does not know when the condition might be met, such as when running sections of code based on user input that will only cease when a user inputs a particular key sequence.

An example of a WHILE loop can be seen in Figure 2.11. This type of WHILE loop is very common, because when opening a file for reading, it is not possible to tell how many lines there are in the file until all those lines are actually read.

```
ALGORITHM readFromFile()
BEGIN
        INPUT fileName

        fileObject ← open filename for reading
        WHILE end of file is not reached DO
                nextLine ← read one line from fileObject
                PRINT nextLine
        ENDWHILE
END
```

FIGURE 2.11 An example of a WHILE loop to read from a file

Key elements of WHILE loops:

- They are used when it is unknown how many times the loop will execute.
- The condition being tested within the WHILE loop must be met at least once for the code within it to be executed.
- If the condition being tested within the WHILE loop is always true, the loop will never terminate; this creates an **infinite loop**.

DO/WHILE loops

A **DO/WHILE loop** is similar to a WHILE loop in that it executes code within the loop for as long as a condition is met.

An example of pseudocode to read the contents of a file using a DO/WHILE loop is shown in Figure 2.12.

```
ALGORITHM readFromFile()
BEGIN
        INPUT fileName

        fileObject ← open filename for reading
        DO
                nextLine ← read one line from fileObject
                PRINT nextLine
        WHILE end of file is not reached
END
```

FIGURE 2.12 An example of a DO/WHILE loop to read from a file

SCHOOL-ASSESSED TASK TRACKER

☐ Project plan	☐ Justification	☐ Analysis	☐ Folio of alternative designs ideas	☐ Usability tests	☐ Evaluation and assessment	☐ Final submission

9780170440943

The difference between a WHILE loop and a DO/WHILE loop is that a WHILE loop may not run if the condition being tested is never true, whereas a DO/WHILE loop always runs at least once. For example, in the pseudocode shown in Figure 2.12, it is assumed that there is at least one line to read in the file that is opened. If this is not the case and this code were to be implemented in a programming language, it would produce a runtime error.

Key elements of DO/WHILE loops:

- They are used when it is unknown how many times the loop will execute.
- The code within the DO/WHILE loop will always execute at least once.
- If the condition being tested within the DO/WHILE loop is always true, the loop will never terminate; this creates an infinite loop.

FOR loops

A **FOR loop** is a section of code that is run a pre-defined number of times. These types of loops are particularly useful to perform an action on every element of an array, or to perform a basic search through a set of elements in a data structure.

FOR loops need three pieces of information to execute. The first is a starting point, the second is the end condition, and the third is a statement called an increment that increases the starting point so that it approaches the end condition.

Most programming languages have a special format for writing FOR loops. For example, in C and C++, a FOR loop is written as shown in Figure 2.13, and in Visual Basic it is written as shown in Figure 2.14.

```
for ( starting_point; end_condition; increment )
{
      statement(s);
}
```

FIGURE 2.13 A FOR loop in the style of the C, C++ and C# programming languages

```
For counter = start_condition to end_condition
      Statement(s)
Next
```

FIGURE 2.14 A FOR loop in the style of the Visual Basic programming language

An example of a FOR loop in pseudocode is shown in Figure 2.15. This FOR loop checks every element of an array to see if a word being searched for is found in the array. Each time the loop is executed, the end condition (i < iNumNames) is tested; the loop will continue to run for as long as this returns true.

```
ALGORITHM checkArray()
BEGIN
       INPUT arrayNames
       INPUT searchTerm

       iNumNames ← length of arrayNames
       FOR i ← 0, i < iNumNames, i ← i + 1 DO
              IF arrayNames[i] = searchTerm THEN
                     PRINT "Found " + searchTerm
              ENDIF
       ENDFOR
END
```

FIGURE 2.15 A FOR loop checking every element of an array

9780170440943

Key elements of FOR loops:

- The loop runs for a set number of times and this must be known beforehand.
- The loop will only execute the code inside it if the end condition is still being met; it may not execute the code at all.
- Unlike WHILE and DO/WHILE loops, it is very rare for a FOR loop to not terminate; this occurs only if the increment and end condition are unrelated. This type of occurrence would be considered a **logic error**.

REPEAT/UNTIL loops

Much like WHILE loops, **REPEAT/UNTIL loops** repeatedly run a source code within the loop, however they differ in the treatment of the condition that terminates the loop. A WHILE loop will run for as long as a condition returns true, whereas a REPEAT/UNTIL loop will run for as long as a condition returns false. Figure 2.16 demonstrates the pseudocode that uses a REPEAT/UNTIL loop to read lines from a file.

```
ALGORITHM readFromFile()
BEGIN
        INPUT fileName

        fileObject ← open filename for reading
        REPEAT
                nextLine ← read one line from fileObject
        UNTIL end of file is reached
END
```

FIGURE 2.16 An example of a REPEAT/UNTIL loop to read from a file

Key elements of REPEAT/UNTIL loops:

- They are used when it is unknown how many times the loop will execute.
- The code within the REPEAT/UNTIL loop will always execute at least once.
- If the condition being tested within the REPEAT/UNTIL loop is always false, the loop will never terminate; this creates an infinite loop.

Functions

A **function** is a sequence of instructions that performs a specific task that has been given a name by a programmer. The code within a function executes an algorithm and typically provides a **return value** as a result. To use a function within source code is to 'call' it. An example of a **function call** can be seen in Figure 2.17; both `useUmbrella()` and `getWet()` are function calls.

```
ALGORITHM checkUmbrellaUsage()
BEGIN
        INPUT isRaining
        INPUT hasUmbrella

        IF isRaining = True THEN
                IF hasUmbrella = True THEN
                        RETURN useUmbrella()
                ELSE
                        RETURN getWet()
                ENDIF
        ENDIF
        RETURN False
END
```

FIGURE 2.17 An example of a function call

Notice that functions in pseudocode use parentheses after the name of the function. This helps distinguish them from variables.

SCHOOL-ASSESSED TASK TRACKER

☐ Project plan	☐ Justification	☐ Analysis	☐ Folio of alternative designs ideas	☐ Usability tests	☐ Evaluation and assessment	☐ Final submission

9780170440943

A function can, but does not need to, provide a return value back to where the function was called.

TABLE 2.4 Function declarations in some popular programming languages

C, C++, C#	int max(numOne, numTwo);
VB, VB.Net	Function max(byVal numOne as Single, byVal numTwo as Single)
Python	def max(numOne, numTwo):
PHP	function max($numOne, $numTwo);
Swift	func max(numOne: Int, numTwo: Int) -> Int

Many programming languages require functions to be declared before they can be defined. Similarly, functions must be defined before they can be used within the source code. For this reason it is a good idea to place all function declarations at the top of your source code, and function definitions near the top as well.

Functions require a **function declaration** that names the function and its **arguments**. In many languages, function declarations must include the return value data type. They can also include an optional reference to the **function visibility**.

Functions must then have a **function definition** written, which simply means that the function must be written. Some languages, such as C and C++, require that the function declaration is written separately, prior to the function definition, whereas other languages, such as Python and VB.Net, include the function declaration as part of the function definition. Once a function is defined and written it can be used throughout the source code.

Almost all programming languages have **built-in functions** that can be used without needing to provide a function declaration or function definition. These functions have been written by the creators of the programming language to execute common sequences of code, such as drawing a widget on a user interface, printing text to a screen, computing mathematical equations such as square roots and powers, or accepting input from a keyboard.

It is best to use a built-in function whenever possible, as they have been tested and are less likely to contain bugs than a user-defined function.

Arguments/parameters

Functions can optionally include variables in their definition. These are known as arguments, or **parameters**, and they act as specific inputs that are 'passed' to the function when the function is called. The data within the arguments passed to a function are assigned temporary variable names as part of the function declaration. This allows the use of **local variables** within a function, avoiding the need to use **global variables** to access data that exists outside of the function.

Many programming languages have two categories of arguments that can be passed to a function: those that are **pass by reference** and those that are **pass by value**. Pass by reference means that the original data being passed into the function can be modified without needing to be 'returned'. Pass by value means that the original data is left unchanged, even if the data in the temporary variable is modified within the function. For languages that only use pass by value in functions, in order to modify the data stored in the original variables that have been passed to the function, the modified data must be returned back to the source code that called the function.

Figure 2.18 demonstrates how to declare a function using pseudocode. Note that the data types of the arguments and the function's return value are defined within a pseudocode comment. The arguments are included as part of the function definition to distinguish them from other types of input.

9780170440943

```
FUNCTION max(numOne, numTwo)
{ Purpose: return max value as an integer }
{ Arguments: numOne and numTwo are integers }
{ Output: integer, the maximum integer }
BEGIN
        IF numOne > numTwo THEN
                RETURN numOne
        ELSE
                RETURN numTwo
END
```

FIGURE 2.18 Representation of a function in pseudocode

Function visibility

Visibility and scope are elements of programming that can be applied not just to functions, but also to variables, methods, classes and events.

Security is an important aspect to consider when writing source code for modern applications. Aside from encryption, there are other ways a software developer can protect access to data within variables. The most common method is to use function visibility to restrict access to functions within applications.

Public

Public visibility of a function means that it is visible both inside the source code or class in which the function exists, and also via external source files, classes and applications.

While a programmer can explicitly refer to a function as public, there is no need to do this in most programming languages, as public visibility is the default visibility of all functions.

Protected

Protected visibility of a function means that it is visible only to a class or extensions of that class. This means that the functions and methods defined as protected within a class can only be used by that class as well as by any classes that inherit the class that contains the protected function.

Private

Private visibility of a function means that it is visible only to a particular class. Unlike protected functions, a private function cannot be used by a class that inherits the class that contains the private function.

Classes

As mentioned in chapter 1, a class is a feature of object-oriented programming that allows a programmer to group together related functions and variables in one place. This acts as a template for creating objects.

In a business application, a programmer could write a 'user' class that contains typical user variables and methods, such as username, password, the ability to log in, and the ability to change user details. They could then use this base 'user' class to create an 'administrator' class that adds other methods, such as the ability to change other users' passwords and user details. This is demonstrated in Figure 2.19, where the Administrator class is inheriting all of the variables and methods from the User class and then adding three more. These two classes, Administrator and User, could then be instantiated into objects within the program to allow for two different types of users to exist. This saves programming time, as the basic user methods do not need to be rewritten when creating a different type of user.

9780170440943

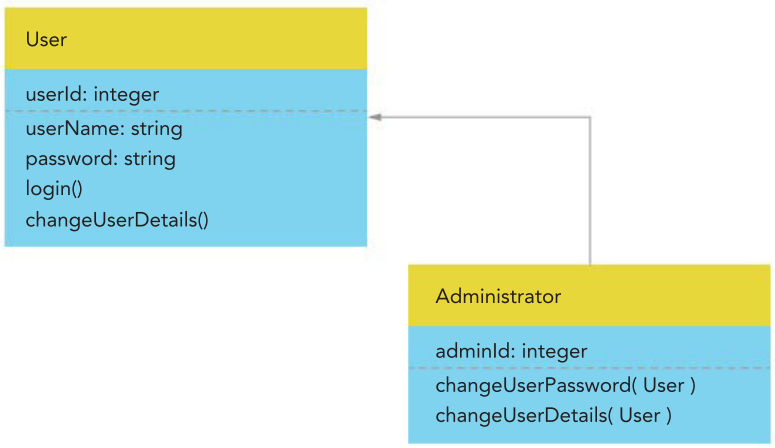


FIGURE 2.19 An example of inheritance

Classes are also useful when security of data is paramount. Much like function visibility, variables inside a class can have **class visibility** and be 'hidden' from other parts of the code. The only way to access the data in those variables is to use a method that has been written with security in mind. These methods may, for example, encrypt or decrypt data or check user permissions before allowing a variable's data to be changed.

Methods

Events can be user-defined or they can be built-in events.

A **method** is a function that exists within a class. A special type of method is an **event**, which is a method that is called when an object's state changes; this means that something has occurred to trigger the event. For example, pressing a button in a user interface object can trigger an event to submit that button.

```
ALGORITHM useMethod()
BEGIN
    mathObj = create new object from math class
    mathObj.addTwoNumbers(4, 1)
END
```

FIGURE 2.20 Pseudocode of an object and a method

Algorithms for sorting

As sorting algorithms are not language specific, almost all programming languages have built-in functions that implement standard sorting algorithms for programmers to use.

Many applications require some method of sorting data so it can be used within a program or by a user of a program. It may seem trivial for a human to place a set of items into sorted order, but to achieve the same result in a computer program requires the use of control structures and repetition of sequences of steps. It is also important that the combination of control structures and sequences are efficient, particularly when a lot of data needs to be sorted.

9780170440943

While there are many different algorithms that can be used to sort data, Software Development students must know about two specific types of sort: selection sort and quicksort.

Selection sort

A simple method of sorting a set of elements into a sorted list is to use selection sort. **Selection sort** involves searching through a whole list, selecting the smallest element it finds, and swapping that element to the front of the list. The searching and swapping continues until the entire list is sorted.

The selection sort algorithm follows these steps:

- Assume the first element is the smallest element.
- Compare the first element to every other element in the array, one by one.
- Each time the element compared to the first element is smaller, swap the first element with the smaller element.
- Compare the (possibly new) first element with the rest of the array.
- Repeat the whole process, starting with the second element in the array.
- Repeat the whole process, starting with the third element in the array.
- Continue repeating the whole process until all elements in the array have been checked.

This algorithm is called selection sort because it repeatedly selects the next-smallest element and swaps it into place.

For example, consider the following list of unsorted numbers:

12	8	31	1

Assume the first element is the smallest element, in this case, 12. Keep track of the index value of this element in a variable, such as one called 'smallest'. For this example, index values start at 1, but in almost all programming languages they begin at 0.

smallest	1

Compare 12 with 8. 8 is smaller, so update the variable 'smallest' with the index value for the number 8.

smallest	2

The number 8 is then compared to 31. As 31 is not smaller than 8, the 'smallest' variable is unchanged.

The number 8 is then compared to 1. 1 is smaller, so update the variable 'smallest' with the index value for the number 1.

smallest	4

THINK ABOUT SOFTWARE DEVELOPMENT 2.1

Take a shuffled deck of cards, and physically sort them. Consider how you chose to sort them. Did you separate suits into four stacks and then sort by face value? Or did you sort by face value and then by suit? What steps did you repeat to finally succeed in having a sorted deck of cards?

SCHOOL-ASSESSED TASK TRACKER

☐ Project plan	☐ Justification	☐ Analysis	☐ Folio of alternative designs ideas	☐ Usability tests	☐ Evaluation and assessment	☐ Final submission

9780170440943

The end of the array has been reached, so the element in the first position of the array is now swapped with the element at the index position stored in the variable 'smallest'.

1	8	31	12

The sort then moves to the next element of the array and uses the index value of this element as the new value for 'smallest'.

smallest	2

The number 8 is the new smallest element, in position two of the array. It is compared with 31, which is larger, so no change is made to the variable 'smallest'. It is then compared to 12, which is also larger, so no change is made to the variable 'smallest'.

1	8	31	12

As the end of the array has been reached, the element referred to by the variable 'smallest' should be swapped with the 8, but as they are the same index values, no swap needs to happen.

1	8	31	12

The sort then moves to the next element of the array (the third element) and uses the index value of this element as the new value for 'smallest'.

smallest	3

The number 31 is the new smallest element, in position three of the array. It is compared with 12, which is smaller, so the variable 'smallest' is updated with the index value of the 12.

smallest	4

As the end of the array has been reached, the element referred to by the variable 'smallest' is swapped with the 31.

1	8	12	31

The sort then moves to the next element of the array (the fourth element). However, this is the last element of the array so there is nothing left to compare it to. The selection sort algorithm is therefore complete and all elements in the array are now sorted.

Figure 2.21 shows the pseudocode for selection sort.

9780170440943

```
ALGORITHM selectionSort(arrElements)
{ Purpose: sorts a list of elements }
{ Input: an array of elements }
{ Index values start at 1 }
{ Output: Array, a sorted array of elements }
BEGIN
      n ← number of items in arrElements
      FOR i ← 1 to n - 1 DO
        { select the smallest item }
        smallest ← i
        { compare smallest to the rest of the array }
        FOR j ← i + 1 to n DO
            IF arrElements[j] < arrElements[smallest] THEN
                { update the index value of smallest }
                smallest ← j
            ENDIF
        ENDFOR
        { the smallest item in the array has been found }
        { so swap it with the current element }
        IF smallest != i THEN
            swap arrElements[smallest] AND arrElements[i]
        ENDIF
      ENDFOR
      RETURN arrElements
END
```

FIGURE 2.21 Pseudocode for selection sort

When converting pseudocode to real code, some of the pseudocode elements may need to be expanded upon. For example, in most programming languages, swapping two elements generally cannot be achieved using a single line of code. Consider the different outputs of Figures 2.22 and 2.23. Only Figure 2.23 will successfully swap the two values, as the x value is lost in Figure 2.22.

```
x ← 20
y ← 10

x ← y
y ← x
PRINT x, y
```

FIGURE 2.22 Incorrectly swapping two values

```
x ← 20
y ← 10

temp ← x
x ← y
y ← temp
PRINT x, y
```

FIGURE 2.23 Correctly swapping two values

THINK ABOUT SOFTWARE DEVELOPMENT 2.2

Convert the pseudocode in Figures 2.22 and 2.23 into the language of your choice. Run both versions to see their outputs.

SCHOOL-ASSESSED TASK TRACKER

☐ Project plan	☐ Justification	☐ Analysis	☐ Folio of alternative designs ideas	☐ Usability tests	☐ Evaluation and assessment	☐ Final submission

9780170440943

```
def selectionSort( aList ):
    for i in range( len( aList ) ):
        smallest = i
        for k in range( i + 1 , len( aList ) ):
            if aList[k] < aList[smallest]:
                smallest = k
        if smallest != i:
            temp = aList[smallest]
            aList[smallest] = aList[i]
            aList[i] = temp
```

FIGURE 2.24 An example of selection sort implemented using Python

Quicksort

Another algorithm used to sort sets of elements is **quicksort**, referred to as a **divide and conquer** algorithm. This is because quicksort is an example of a recursive sort, which means that it partitions the items that need to be sorted into smaller and smaller sets and passes those sets back into itself; thus, it 'divides' the array into smaller and smaller pieces until it can 'conquer' the array and sort it. This allows items to be sorted very quickly. In almost all cases, quicksort is more efficient than selection sort.

The quicksort algorithm follows these steps:

- Unless the array contains only a single element, complete the following steps:
  - Select an element from the array at random – this is referred to as the pivot. Often, this 'random' element is the last element in the array.
  - Check each other element in the array and reorder it so that all elements with values less than the pivot come before it, while all elements with values greater than the pivot come after it (equal values can go either way). This involves swapping, much like in selection sort, but the elements are not sorted based on anything else except the pivot when they are swapped.
  - Take all of the elements that are less than the pivot (all the elements to the left of the pivot) and repeat the process of quicksort on these elements, selecting a new pivot.
  - Take all of the elements that are greater than the pivot (all of the elements to the right of the pivot) and repeat the process of quicksort on these elements, selecting a new pivot.
- If the array contains only a single element, return just that element.

For example, consider the following list of unsorted numbers:

12	8	31	1	77	75	18

Select the first element as the pivot and store its index value in a variable, 'pivot'. Two more variables need to be made, one storing the first index value, and the other storing the last index value. At the beginning of the quicksort algorithm, there should therefore be three variables: for this example, they are called pivot, low and high. Assuming index values begin at 1, the index value of pivot is 7, low is 1 and high is 7.

Begin iterating through the list, comparing each value to the value stored at the pivot. The first element checked is 12, which is smaller than the pivot's value of 18. This means the first element should be swapped with the element at the 'low' index value. In this instance, the 'low' index value is the same index as the first element, so no

9780170440943

swap occurs. After this, the 'low' index value is incremented by 1, and now holds the value of 2.

12	8	31	1	77	75	18

The next element checked is 8, which is smaller than 18. This means that this element must be swapped with the element at the 'low' index value. In this instance, the 'low' index value of 2 is the same as the second element, so no swap occurs. The index value of 'low' is then incremented by 1, and now holds the value of 3.

12	8	31	1	77	75	18

The next element checked is 31, which is larger than 18. This element is left alone.

12	8	31	1	77	75	18

The next element checked is 1, which is smaller than 18. This means that this element must be swapped with the element at the 'low' index value; 1 and 31 are therefore swapped. The index value of 'low' is then incremented by 1, and now holds the value of 4.

12	8	1	31	77	75	18

The next element checked is 77, which is larger than 18. This element is left alone. The same occurs with 75.

Once the algorithm reaches the last element, 18, it is swapped with the element at the index value of 'low'. In this case, this is the element at the fourth index value, 31.

12	8	1	18	77	75	31

The list is now partitioned so that every number less than the pivot (18) is to the left of it, and every number that is greater is to the right of it. The algorithm is run again on each side.

Left side:

12	8	1

Pivot: 3 (value: 1)
Low: 1 (value: 12)
High: 3 (value: 1)

After the first pass, no swaps are made as 12 is greater than 1:

12	8	1

Pivot: 3 (value: 1)
Low: 1 (value: 12)
High: 3 (value: 1)

9780170440943

After the second pass, no swaps are made as 8 is greater than 1. There are no more passes to make, so the 'low' and 'high' index values are swapped:

1	8	12

The list is now partitioned so that every number less than the pivot (1) is to the left of it, and every number that is greater is to the right of it. The algorithm is run again on each side, even though one of the sides (the left side) is empty. These steps are not shown here, as this particular sub-list is already in sorted order.

Right side:

77	75	31

Pivot: 3 (value: 31)
Low: 1 (value: 77)
High: 3 (value: 31)

After the first pass, no swaps are made as 77 is greater than 31:

77	75	31

Pivot: 3 (value: 31)
Low: 1 (value: 77)
High: 3 (value: 31)

After the second pass, no swaps are made as 75 is greater than 31. There are no more passes to make, so the 'low' and 'high' index values are swapped:

31	75	77

The list is now partitioned so that every number less than the pivot (31) is to the left of it, and every number that is greater is to the right of it. The algorithm is run again on each side, even though one side (the left side) is empty. These steps are not shown here, as this particular sub-list is already in sorted order.

Once all sub-lists have been processed through the quicksort algorithm, the array that is left is in sorted order:

1	8	12	18	31	75	77

Figure 2.25 shows the pseudocode for quicksort.

While quicksort is algorithmically complex for a human brain to understand, **recursive algorithms** are extremely fast for a computer to process.

9780170440943

```
ALGORITHM quickSort(arrElements, low, high)
{ Purpose: sorts a list of elements }
{ Inputs: an array of elements, two integers representing }
{ the first last element in the array }
{ Index values start at 1 }
BEGIN
       IF low < high THEN
           { run the partition algorithm to know where }
           { to split the array }
           split ← partition(arrElements, low, high)

           { run quicksort on the left side }
           quickSort(arrElements, low, split-1)

           { run quicksort on the right side }
           quickSort(arrElements, split+1, high)
       ENDIF
END

ALGORITHM partition(arrElements, low, high)
{ Purpose: to split an array into two based on a pivot, }
{           where the left side contains values less than }
{           the pivot and the right side contains value }
{           greater than the pivot }
{ Inputs: an array of elements, two integers representing }
{           the first last element in the array }
{ Index values start at 1 }
{ Output: integer, index value of the partition point }
BEGIN
       pivot ← arrElements[high]

       FOR i ← low to high DO
           IF arrElements[i] ← pivot THEN
               IF low != i THEN
                   swap arrElements[low] and arrElements[i]
               ENDIF
               low = low + 1
           ENDIF
       ENDFOR
       swap arrElements[low] and arrElements[high]
       RETURN low
END
```

FIGURE 2.25
Pseudocode for quicksort

Algorithms for searching

As with sorting, many applications require some method of searching through data to find a particular item within a set of items. It is important that these searches are efficient, particularly in the modern era of 'big data'.

While there are many different algorithms that can be used to search data, Software Development students must know about two specific types of searches: linear search and binary search.

Linear search

A **linear search** is the simplest type of search. This search involves checking every element in the list, from first to last, when searching for a particular element.

SCHOOL-ASSESSED TASK TRACKER

☐ Project plan	☐ Justification	☐ Analysis	☐ Folio of alternative designs ideas	☐ Usability tests	☐ Evaluation and assessment	☐ Final submission

9780170440943

For example, consider the following list of unsorted numbers, where the number being searched for is 77:

12	8	31	1	77	75	18

A linear search is typically performed using a FOR loop if the size of the list is known. The first step of the FOR loop will check the first index value, which is 12. 12 is compared with the search input, 77, and this returns false. As a result, the FOR loop will iterate again, checking the next index value in the list, 8. This repeats until it reaches the 77, where the linear search returns true, as it has found the value. Linear searches will execute until the end of the list is reached or the search item is found.

Figure 2.26 shows the pseudocode for a linear search. Linear searches are useful when the elements are not sorted and the number of elements to search through is small.

```
ALGORITHM linearSearch( )
{ Purpose: searches through a list of elements }
{ Output: Boolean, True if item found, False if not }
BEGIN
        Input searchList, searchItem
        found ← FALSE
        FOR eachItem in the searchList DO
          IF eachItem = searchItem THEN
            found ← TRUE
            BREAK {exit loop once found}
          ENDIF
        ENDFOR
        RETURN found
END
```

FIGURE 2.26 Pseudocode for a linear search

While linear searches are the simplest to implement in almost every programming language, they are also extremely inefficient. This is not a concern if the number of elements being checked is relatively small, but it becomes problematic as the number of elements increases. Consider a search of 6 billion records where the record being searched for is not there. This is referred to as a 'worst-case' scenario, as all 6 billion records would need to be checked to confirm that the record was not in the set.

Binary search

A **binary search** is more efficient than a linear search. Binary search is similar to quicksort in that it is also a recursive algorithm, but instead of being divide and conquer, it is a **decrease and conquer** algorithm, as it is able to discard half of the data being searched through at each iteration of the algorithm. This makes it an extremely efficient method of searching.

Binary search relies on the data it is searching through being sorted. It works by selecting an element from the very middle of the data set being searched and checking it against the search item. If it matches, the search halts. If it doesn't match, it will search the data to the left of the element in the middle if it is less than that element, otherwise it will search the data to the right of it. This process is repeated until the item is found or there are no more elements to search through.

For example, consider the following list of sorted numbers, where the number being searched for is 77:

1	8	12	18	31	75	77

9780170440943

A binary search will begin by checking the length of the list. If it is greater than one, it will then find the index value of the middle of the list by dividing the length of the list by 2. As the list has 7 elements, dividing by 2 produces a fraction, which must be rounded up or down. It doesn't matter which way it is rounded, as long as the rounding is consistent. For this example, the binary search will round up, therefore the index value that is checked first is at index 4. 18 is compared to the number being searched for and does not match. The binary search will then check to see if 18 is greater than or less than 77. As it is less than 77, the search will discard every element to the left of the 18, inclusive. The rest of the list is passed back into the binary search to be searched again.

31	75	77

There are three elements in the list, so the mid-point is calculated again; this time it is index value 2. The number 75 is compared to 77 and does not match; it is also less than 77 so it is discarded, as is every element to the left of it. The rest of the list is passed back into the binary search to be searched again.

77

There is only one item left in the list, so no mid-point needs to be calculated. The item is compared to the search item and it matches, so the search item is found.

Figure 2.27 shows the pseudocode for a binary search. Binary searches are useful when there are large amounts of elements to search through, but can only be used if those elements are sorted. This means that you need to consider the time it takes to *sort the list* as well as *search the list* when considering the efficiency of binary search.

```
ALGORITHM binarySearch(arrayList, searchItem)
{ Purpose: searches through a list of elements }
{ Inputs: an array of elements to be searched }
{          and the item being searched for }
{ Output: Boolean, True if item found, False if not }
BEGIN
       found ← FALSE
       iLen ← the length of arrayList
       midP ← the middle index value of arrayList
       IF searchItem = arrayList[midP] THEN
          found ← TRUE
       ELSEIF iLen > 1 THEN
          IF searchItem < arrayList[midP] THEN
             low ← first index value of arrayList
             RETURN binarySearch(arrayList[low to midP-1],
                                 searchItem)
          ELSEIF searchItem > arrayList[midP] THEN
             high ← iLen
             RETURN binarySearch(arrayList[midP to high],
                                 searchItem)
          ENDIF
       ENDIF
       RETURN found
END
```

FIGURE 2.27 Pseudocode for a binary search

9780170440943

That being said, the efficiency of a binary search versus a linear search is not a small comparison. Consider the search discussed earlier, where there are 6 billion records and the record being searched for is not there. Recall that in a worst-case scenario, if a linear search was used, all 6 billion records would need to be checked to confirm that the record was not there. If a binary search was used, only 33 items at most would need to be checked to come to the same conclusion. The difference in speed of a linear search and binary search in a worst-case scenario can be seen in the graph shown in Figure 2.28.

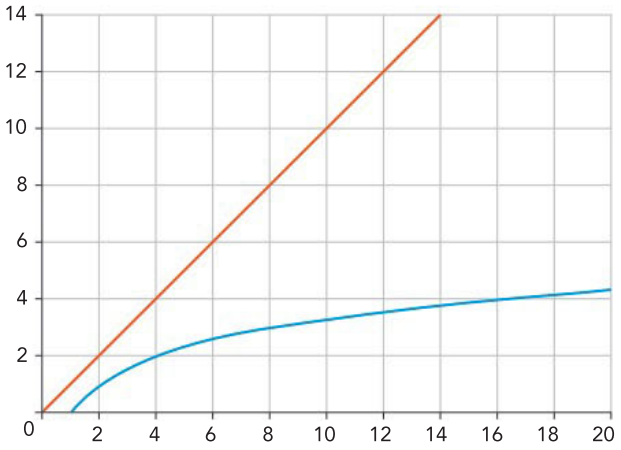


FIGURE 2.28 Worst-case linear search (red) vs. worst-case binary search (blue)

Efficiency of algorithms

When more than one algorithm could achieve a particular purpose, it is important to consider the efficiency of each algorithm. Some algorithms perform better in certain contexts than others, and some are simply better than others overall.

The mathematical notation used to describe the efficiency of algorithms is referred to as 'Big O notation', and while it is not explicitly in the Software Development course, it is useful to know the Big O notation for algorithms in order to make an informed decision on which one to use. Big O notation is expressed using a capital O and then enclosing in brackets the maximum amount of time it would take for an algorithm to finish, expressed in terms of the number of items being processed by the algorithm. For example, if a sorting algorithm used a single FOR loop to sort all of the elements, n, in a list, it would be expressed as running in $O(n)$ time. If a nested FOR loop were used to sort the same list, where one FOR loop runs inside of another FOR loop, it would be expressed as running in $O(n^2)$ time. Divide and conquer algorithms generally involve logarithmic time.

When measuring algorithm efficiency, there are three scenarios that are considered: **best case**, **average case** and **worst case**. The best-case scenario for many sorting algorithms, for example, is for the set of elements to already be sorted. The worst-case scenario for many of those same sorting algorithms would be for the elements to be in reverse-sorted order. The average-case scenario for a sorting algorithm would be when the elements are not in any particular order. Generally, programmers consider the average case of an algorithm when deciding which one to use, and then consider the worst-case scenario if the average cases are equivalent. They may also consider the worst-case scenario to make sure that it is not considerably slower than an alternative algorithm.

9780170440943

For example, on average, a linear search will check approximately half $\left(\frac{n}{2}\right)$ the number of items before it finds the item being searched for. A binary search will check approximately $\log_2 n$ items, which is more efficient. In a worst-case scenario where an item is not present, a linear search must check n items, whereas a binary search divides the list in two each time it iterates through it, which means checking $\log_2 n$ items. This means that a binary search is faster than a linear search in both average-case and worst-case scenarios.

Only average-case and worst-case scenarios are typically considered because it is often not helpful to consider the best-case scenario of an algorithm. Data is rarely in a format that would allow for a best-case scenario to occur, so best-case scenarios are very rare. For example, in a best-case scenario linear search, the item being searched for would be the first item in the list. This is not something that would occur reliably enough to matter when deciding which algorithm to use.

For the two sorting algorithms and two searching algorithms used in Software Development, the average case and worst case are shown in Table 2.5.

TABLE 2.5 Average case and worst case of Software Development algorithms

Average case		
	Selection sort	$O(n^2)$
	Quicksort	$O(n \log_2 n)$
	Linear search	$O\left(\frac{n}{2}\right)$
	Binary search	$O(\log_2 n)$
Worst case		
	Selection sort	$O(n^2)$
	Quicksort	$O(n^2)$
	Linear search	$O(n)$
	Binary search	$O(\log_2 n)$

As you can see, in a worst-case scenario, quicksort combined with binary search is no worse than selection sort combined with binary search. However, once the average case is considered, the most efficient method of sorting and searching is to combine a quicksort with a binary search.

Validation techniques

Validation is the process of checking that input data are reasonable. Validation does not and cannot check that inputs are accurate.

Existence checking

An **existence check** checks whether a value has been entered at all. This is particularly useful to ensure that all required fields in a form have been completed before saving the contents of those fields to a file.

9780170440943

Type checking

A **type check** is a useful method of confirming that the values entered into fields are of the expected data type. It will confirm if the wrong type of data have been entered in fields, such as if strings are entered into fields that expect only integer values.

When implementing type checking, it is important to consider how inputs from a user are processed by the selected programming language. For example, in Python, unless explicitly handled otherwise, all inputs are treated as strings, even if they are numeric. In languages such as these, there typically exists methods for checking type, such as the process of **casting** the data as a particular data type to see if it remains valid.

Range checking

A **range check** checks that data are within acceptable limits or come from a range of acceptable values. For example, students enrolling in kindergarten must be between the ages of 3 and 6 years (acceptable limits). As another example, the product size must be small, medium or large (acceptable values).

Checking that modules meet design specifications

When building an application, one of the most important steps is to check that each completed module meets the design specifications.

When checking to see if a module meets design specifications, it is important to make sure it performs as expected with appropriate inputs, that it is usable and efficient, and that it achieves what was specified in the design. This often involves internal testing, where programmers tests the program themselves; external testing, where quality assurance testers test the program using test cases based on the design specifications; and client testing, where clients participate in walk-throughs and reviews of the software to confirm that it is what they specified in the design stage.

In VCE Software Development, modules are checked to ensure they meet design specifications mostly through internal testing methods. An important aspect of this is to make sure that the modules that have been built are as bug-free as possible. This is achieved by completing appropriate testing activities, such as establishing test cases that determine test data and **expected results**, conducting tests, recording results and then correcting any errors.

One method used in the testing process is **debugging**, which is a testing method that includes finding errors through the construction of trace tables and testing the system using test data.

Debugging

Different types of errors can occur throughout the development of a software solution. Each error can be categorised as a syntax error, runtime error or logic error.

Syntax errors

Each programming language has a defined syntax, which is a set of rules that defines which symbols and characters can be used to write source code. **Syntax errors** occur while writing

code and are typically fixed immediately, as they prevent code from compiling. These are errors such as missing brackets or semi-colons, not passing the correct number of arguments into a function, or not putting quotes around characters and strings. Most compilers and interpreters will indicate on which line a syntax error occurs, which allows a programmer to find the error easily.

Runtime errors

Unlike a syntax error, a software module with a **runtime error** will compile without any error notifications. It is not until the program is run and used that a runtime error can appear. These types of errors often result in the program crashing or printing error messages. For example, one type of runtime error is a **memory leak**, where a program continually uses more and more RAM while the program is running, such as when an infinite loop occurs. Other common runtime errors include **divide-by-zero errors**, opening non-existent files for reading, calling functions that do not exist, or validation errors that have not been handled within the source code.

Runtime errors are often found during the development stage, but software can, and does, get released with the possibility of runtime errors still occurring. This is because these errors tend to occur only when certain conditions are met, or when unexpected inputs are entered into the software. Once a runtime error is found in a distributed piece of software, many software companies release **patches** and software updates to correct the error.

Logic/semantic errors

Logic errors, also known as semantic errors, occur when the logic of a software program fails. This means that the source code is syntactically correct but the software solution does not produce the expected output when run. In this case, the output is often unintended, undesirable or incorrect. For example, a function written to return the square root of a number may instead return the square of a number. Similarly, a function in an air-conditioning unit that checks to see if the temperature in a room is greater than 30 degrees before turning itself on may activate the air-conditioning unit at 29 degrees instead.

Logic errors can be very difficult to find, as there is nothing within a compiler or interpreter that will tell a programmer on which line a logic error appears. It is often up to the programmer as a human to construct test data and trace tables in order to track down the error manually.

Test data

In order to systematically test that a module works, appropriate **test data** must be used to write a test case. A **test case** is a set of steps that a tester uses to determine if the element being tested works correctly. It involves selecting test data, writing testing procedures and determining expected results. It is particularly important that appropriate test data is selected so that test cases can be run. At a minimum, the selected test data must ensure full coverage of the algorithm when test cases are run. This means that all paths of all control structures are tested fully.

There are four main types of test data. The first involves validation test data that tests the validation techniques that have been included in the module. Data must therefore be selected to test any instances of existence checking, type checking and range checking that have been included in the source code. At a minimum, this should involve selecting test data

9780170440943

that checks for both valid and invalid inputs. For example, if checking for a valid age integer being input by a user, a valid input would be a positive integer, likely in the range of 0 and 120. An invalid input would be a negative integer, such as –1.

The second type of test data is data that will test all conditions, including the operators within those conditions. This may involve the construction of truth tables to help choose appropriate test data. It may also involve selecting appropriate boundary values to ensure full coverage of the operators included in the conditions.

The third type of test data involves data that will test all iterations, including any operators within those iterations. This may involve selecting or creating data that will ensure all conditions of those iterations are met. For example, if a WHILE loop is designed to iterate over each line of a file, test data would involve testing the following:

- when the file does not exist
- when the file exists but is empty
- when the file exists and has one line to read
- when the file exists and has more than one line to read.

The fourth type of test data involves creating test cases that will attempt to cover all of the functionality within the system from the perspective of a user. This typically involves testing the Graphical User Interface (GUI) to ensure that each screen within the application can be accessed correctly. Typically, test cases are written based on design documents such as mock-ups, storyboards and/or site maps, to ensure that the order and sequence of user interface elements are correct. This type of testing can be the most time-consuming, particularly if the application allows a user to take multiple paths to access particular screens. Developers often write formal test cases that allow for automated software testing programs to be used to run these tests so they do not need to be performed manually. These test cases can be written to simulate the actions of a user navigating through the software solution.

Boundary values

Testing for **boundary values** involves selecting test data that will test the 'boundaries' of any condition or iteration within the code; that is, the maximum and minimum values available for any given input. Boundary values are particularly relevant for algorithms that use range checks. The general rule for boundary testing is that at each boundary, test data should be selected to test inside the boundary and outside the boundary.

For example, imagine a software solution designed for a Scout group that would only allow users to join if they are in the 'Venturer' age group (15–17 years). The boundary values for testing this can be determined by considering the type of data that is being collected. As this example involves testing for age, valid integers begin at 0 and are unlikely to go beyond 120. Testing for valid ages would therefore require testing an age that is below the age group, within the age group, and above the age group.

The ages that are in range can be represented on a continuum, as shown below.

0	1	2	3	...	14	15	16	17	18	19	...
						↑	↑	↑			

There are two boundaries for this particular test. One boundary exists at age 15 and the other exists at age 17. The pseudocode that demonstrates these can be seen in Figure 2.29.

```
ALGORITHM checkIsVenturer(age)
{ Purpose: to check that a user is a Venturer based on age }
{ Inputs: age, as an integer }
{ Output: Boolean, True if age is valid, False if not }
BEGIN
      IF age > 14 AND age < 18 THEN
            RETURN True
      ELSE
            RETURN False
      ENDIF
END
```

FIGURE 2.29 Pseudocode to check an age range

In order to fully test this algorithm, the test data that must be selected are:

Age	Reason	Expected result
14	One year below the lower boundary of the valid range	False
15	The lower boundary of the valid range	True
17	The upper boundary of the valid range	True
18	One year above the upper boundary of the valid range	False

When selecting above and below a boundary, test data should be in the smallest increment possible given the context. In this case it is 1 year, but if an algorithm were to test for a price range, for example, the smallest increment would be 0.01. Similarly, if testing an algorithm that uses boundaries involving hours or minutes in a day, the smallest increment would typically be 1 minute, thus boundaries at the hour would be 59, 0 and 1.

```
ALGORITHM abstractBoundaries()
BEGIN
      INPUT a, b
      IF a < b THEN
            PRINT "Condition met."
      ELSE
            PRINT "Condition not met."
      ENDIF
END
```

FIGURE 2.30 An abstract set of boundary conditions

For a more abstract example, the pseudocode in Figure 2.30 would require the following boundary values to be tested.

a	b	Reason	Expected result
a	a+1	a is guaranteed to be less than b	Condition met
a	a	a is guaranteed to not be less than b	Condition not met

9780170440943

For more complex algorithms, more test data needs to be selected. This is when it can be very useful to construct a truth table to determine what that test data should be.

Consider the pseudocode example testing for multiple conditions in Figure 2.31.

```
ALGORITHM multiConditions()
BEGIN
        INPUT a, b, c, d
        IF a < b AND (b < c OR c < d) THEN
              PRINT "Condition met."
        ELSE
              PRINT "Condition not met."
        ENDIF
END
```

FIGURE 2.31 A complex conditional with more than one logical expression

Consider also the truth table constructed for this algorithm, shown in Table 2.6.

TABLE 2.6 Truth table for Figure 2.31

a < b	b < c	c < d	b < c OR c < d	a < b AND (b < c OR c < d)
True	True	True	True	True
True	True	False	True	True
True	False	True	True	True
True	False	False	False	False
False	True	True	True	False
False	True	False	True	False
False	False	True	True	False
False	False	False	False	False

The test data required to fully test the conditions in Figure 2.31 must test the conditions shown on each line of the truth table. For example, the second line of the truth table requires that the condition c < d is not met, so a boundary test must be performed where c is not less than d (e.g. c = d).

For this pseudocode, at least eight test data elements must be written to test the conditional statement fully.

It can be useful to use an algebraic expression to map out which values are needed for a test condition to be met. For example, using the example from Figure 2.31 and Table 2.6, the first line of the truth table can be interpreted algebraically in this way:

a < b	b < c	c < d	b < c OR c < d	a < b AND (b < c OR c < d)
a = b – 1	b = c – 1	c = d – 1	True	True

This is because each condition can be guaranteed to be true if the algebraic conditions are met.

9780170440943

The algebraic expressions can be further simplified:

a < b	b < c	c < d	b < c OR c < d	a < b AND (b < c OR c < d)
a = b – 1	b = (d – 1) – 1	c = d – 1	True	True

And again:

a < b	b < c	c < d	b < c OR c < d	a < b AND (b < c OR c < d)
a = ((d – 1) – 1) – 1	b = (d – 1) – 1	c = d – 1	True	True

And finally:

a < b	b < c	c < d	b < c OR c < d	a < b AND (b < c OR c < d)
a = d – 3	b = d – 2	c = d – 1	True	True

Therefore, only a value for d needs to be chosen, as the values for a, b and c can all be determined from this value.

It is important that test data is selected systematically rather than in an ad-hoc manner, otherwise the source code cannot be guaranteed to be logically correct throughout all of its algorithms. This results in a considerably large set of test data to be used to test the system, which is why many software development companies use automated tools to construct and run tests on source code.

Trace tables

To prevent logic errors occurring, programmers often construct **trace tables** to validate the logic of the algorithms used in their source code. Trace tables simulate the execution of a program, referred to as the **flow of execution**. Given test data, each processing feature within an algorithm is executed, step by step, and, based on the test data, the values of the variables that change within that algorithm are tracked to ensure that the logic within the algorithm is correct. This systematic method of tracking the execution of code allows for the thorough testing of even the most complex of algorithms.

As an example, consider the pseudocode in Figure 2.32. This algorithm uses a WHILE loop control structure to print values until a condition is met. The trace table to represent the flow of execution of this pseudocode is in Table 2.7.

```
ALGORITHM sampleWhileLoop()
BEGIN
        x ← 0
        y ← 0
        WHILE x < 32 DO
            x ← x + 8
            y ← y + 4
        ENDWHILE
        PRINT x, y
END
```

FIGURE 2.32 Pseudocode example of a WHILE loop

9780170440943

TABLE 2.7 Trace table for the pseudocode in Figure 2.32

Step	Statement	x	y	output
1	x ← 0	0		
2	y ← 0	0	0	
3	**While** x < 32 **Do**	0	0	
4	x ← x + 8	8	0	
5	y ← y + 4	8	4	
6	**While** x < 32 **Do**	8	4	
7	x ← x + 8	16	4	
8	y ← y + 4	16	8	
9	**While** x < 32 **Do**	16	8	
10	x ← x + 8	24	8	
11	y ← y + 4	24	12	
12	**While** x < 32 **Do**	24	12	
13	x ← x + 8	32	12	
14	y ← y + 4	32	16	
15	**While** x < 32 **Do**	32	16	
16	**EndWhile**	32	16	
17	**Print** x, y	32	16	32, 16

Another example, where a logic error exists within an algorithm, can be seen with the algorithm in Figure 2.33.

```
ALGORITHM applyDiscount()
BEGIN
        INPUT fullPrice
        INPUT discPerc

        discPrice ← fullPrice - discPerc
        RETURN discPrice
END
```

FIGURE 2.33 Pseudocode containing a logic error

As this algorithm requires input from a user, the trace table can only be constructed using test data. For the purposes of this example, the test data is as follows, with the flow of execution being tested twice, with two sets of inputs, and the expected discounted price being listed with the test data.

TABLE 2.8 Test data for pseudocode in Figure 2.33

fullPrice	discPerc	Expected result
20.00	5	19.00
50.00	50	25.00

9780170440943

TABLE 2.9 Trace tables for the pseudocode in Figure 2.33

Step	Statement	fullPrice	discPerc	discPrice	output
1	**Input** fullPrice	20			
2	**Input** discPerc	20	5		
3	discPrice ← fullPrice – discPerc	20	5	15.00	
4	**return** discPrice	20	5	15.00	15.00

Step	Statement	fullPrice	discPerc	discPrice	output
1	**Input** fullPrice	50			
2	**Input** discPerc	50	50		
3	discPrice ← fullPrice – discPerc	50	50	0.00	
4	**return** discPrice	50	50	0.00	0.00

After completing the trace tables, the test data can be completed to show the following results.

TABLE 2.10 Completed test data for pseudocode in Figure 2.33

fullPrice	discPerc	Expected result	Actual result
20.00	5	19.00	15.00
50.00	50	25.00	0.00

The actual results from the trace table and the test data make it clear that the algorithm is not calculating a correct discounted price given the inputs it is receiving; it is merely subtracting the discount percentage as if it were a dollar value to be discounted. The corrected algorithm can be seen in Figure 2.34.

```
ALGORITHM applyDiscount()
BEGIN
        INPUT fullPrice
        INPUT discPerc

        discAmount ← fullPrice * (discPerc / 100)
        discPrice ← fullPrice - discAmount
        RETURN discPrice
END
```

FIGURE 2.34 Discount pseudocode after the logic error is fixed

Trace tables were traditionally produced manually by a programmer to test code and check for logic errors. Many programming languages today, however, have **integrated developer environments** (IDEs) that allow a programmer to trace the flow of execution through a debugger built into the IDE, so that they do not need to construct trace tables by hand.

9780170440943

2 CHAPTER SUMMARY

Essential terms

alternative execution code that is run if a condition is not met

arguments specific inputs passed into a function that act as local, temporary variables

average case the time it takes to run an algorithm, on average

best case the best time it can take to run an algorithm

binary search a decrease and conquer algorithm that repeatedly halves a sorted search space until an element is found or not found

boolean a data type that holds the values of true or false

boundary values the maximum and minimum edge values possible for a given input

built-in functions functions that have been written by the creators of the programming language to execute common sequences of code

casting converting a variable from one data type to another, such as converting a string to an integer

chained conditional a conditional statement that handles more than one possible conditional outcome

class visibility the accessibility of a class from source code; public, private or protected

compiler a program that turns source code into machine language that can be executed by a computer processor

conditional statement a control structure that allows a programmer to write lines of code that are only run when a particular requirement is met

debugging identifying and removing errors from computer software

decrease and conquer to recursively reduce a problem to two or more smaller instances of the same problem until the problem can be solved

definition an instruction that assigns a value to a variable

divide and conquer to recursively break down a problem into two or more sub-problems of the same type until they are simple enough to solve on their own; the solved problems are then combined to provide a final solution

divide-by-zero error an error occurring when an arithmetic equation is attempting to divide by 0

DO/WHILE loop an iteration over a set of instructions, conditions and/or iterations that is repeated for as long as a condition is met; it is always run at least once

event a special type of method that is called when an object's state changes

existence check test to see if a value has been entered as input or not

expected results the output expected from an algorithm, assuming it is logically correct

flow of execution the order in which instructions, conditions and iterations are executed or evaluated

FOR loop an iteration over a set of instructions that is repeated a set number of times

function a sequence of related code that has been given a name that can be called from other points in the source code

function call to execute the contents of a function

function declaration to name a function and its arguments

function definition to define (write) the contents of a function

function visibility the accessibility of a function from source code; public, private or protected

9780170440943

global variables variables that are defined outside any function and can be accessed by all functions throughout the source code

infinite loop an iteration that will never reach the condition upon which it can terminate

instruction a unit of code that can be executed by a compiler or interpreter

integrated developer environment (IDE) software that provides tools to aid in programming, such as source code editing, syntax highlighting, code completion, debugging aids, or tools to help construct a user interface

interpreter a computer program that directly executes source code without needing to have it compiled beforehand

linear search a search that checks every element in a list, from first to last, when searching for a particular element

local variables variables that are defined inside a function that can only be accessed by that function

logic error when source code is syntactically correct but contains an error resulting in unintended, undesirable or incorrect output

logical operator a boolean operator used to combine expressions, such as AND, OR

memory leak a failure of a program to release memory that is no longer needed, causing impaired performance, application failure and/or system failure

method an action an object can carry out; e.g. window.refresh, golfClub.swing

nested conditional when a condition contains one or more additional conditions within its structure

parameters see **arguments**

pass by reference to pass data into a function as an argument so that it can be modified without needing to be returned

pass by value to pass data into a function as an argument so that it cannot be modified without needing to be returned

patches sets of changes to a software application designed to update or fix it

pointer a variable that stores the memory position of another variable's value

quicksort a divide and conquer algorithm that sorts a set of data by recursively partitioning and sorting smaller and smaller sets of that data

range check tests to see if a value is within a given range of acceptable values

recursive algorithm an algorithm that calls itself with smaller or simpler sets of values until a solution can be found

REPEAT/UNTIL loop an iteration over a set of instructions that is repeated for as long as a condition is not met; it will always execute at least once

return value a value or set of values that is passed back to the origin of a calling function, often to be assigned to a variable, used in an equation, or tested within a conditional statement

runtime error an error that occurs while a program is running

selection sort the process of selecting and swapping elements within a list until the entire list is sorted

sequence a set of instructions that executes line by line in the order that it is written

statement a single line of code that, when executed, performs a single action

switch/case a conditional statement that handles more than one possible conditional outcome

syntax error often a typographical error in source code that violates the set of rules that define a programming language

test case a set of steps that a tester uses to determine if the element being tested works correctly, often outlining test data, testing procedures and expected results

test data data that has been specifically identified to be used in a test case

trace table a table used to test an algorithm, typically by hand, to ensure that no logic errors occur

truth table a table used to represent all of the combinations of values for inputs and their outputs, typically used to test conditional statements

type check tests to see if a value is of the specified data type or structure

WHILE loop an iteration over a set of instructions that is repeated for as long as a condition is met

worst case the longest amount of time it can take to run an algorithm

9780170440943

2 CHAPTER SUMMARY

Important facts

1. **Variables** are references to stored data so they can be used within a program.
2. **Instructions** are units of code that can be executed by a compiler or interpreter.
3. **Definitions and statements** are two types of instructions.
4. **Control structures** typically involve **sequences, conditions** and **iterations**.
5. **Sequences** are sets of instructions that execute line by line.
6. **Conditional statements** are control structures that will execute only if a particular requirement is met.
7. **Conditionals** allow for: alternative execution, more than one logical expression, chaining, nesting and switching.
8. **Iterations** are repetitive loops that repeatedly run sections of code until a condition is met or not met.
9. There are four types of iterations: **WHILE**, **DO/WHILE**, **FOR** and **REPEAT/UNTIL**.
10. **Functions** are sequences of related code that have been named by a programmer.
11. Functions contain arguments, also known as **parameters**, that act as local variables to the function.
12. Functions can have visibility within source code, so that they can be accessed by every other function, some other functions, or no other functions.
13. **Classes** allow programmers to group related functions and variables together.
14. Classes can be instantiated to create objects.
15. **Methods** are functions that exist inside classes and objects.
16. **Events** are special types of methods that perform a sequence of code when an action occurs.
17. Two algorithms used to sort data are **selection sort** and **quicksort**.
18. **Selection sort** repeatedly selects the next smallest element from a set of elements and swaps it into its correct position until all elements are sorted.
19. **Quicksort** repeatedly partitions elements into smaller and smaller sets in order to produce a final sorted list.
20. Two algorithms used to search through data are **linear search** and **binary search**.
21. **Linear search** checks every element in a list, from first to last, when searching for a particular element.
22. **Binary search** repeatedly divides and discards half of the elements in a list while searching for an element until the element is found or it is determined to not be in the list.
23. It is important that algorithms are as **efficient** as possible.
24. **Algorithm efficiency** can be described in terms of **best case**, **average case** and **worst case**.
25. **Validation techniques** include existence checks, type checks and range checks.
26. **Existence checks** test to see if any value has been entered as input.
27. **Type checks** test to see if the input entered is of the correct data type.
28. **Range checks** test to see if the input entered is within a range of acceptable values.

9780170440943

29 **Debugging** is the process by which software errors are found and fixed.

30 **Syntax errors** occur when the grammatical rules of a programming language have not been followed.

31 **Runtime errors** occur while the program is being used.

32 **Logic errors** occur when the expected output of an algorithm does not match the actual output, but no runtime errors occur.

33 **Test data** must be carefully selected in order to test all aspects of an algorithm.

34 **Truth tables** can be useful when selecting test data to use.

35 **Test cases** should be created to test all boundary values.

36 **Trace tables** simulate the flow of execution of a program and allow a programmer to manually detect logic errors.

9780170440943

TEST YOUR KNOWLEDGE

Review quiz

Variables and instructions

1 Which of the following are most likely to be variables?

- **a** getUmbrella()
- **b** isFound
- **c** customer.firstName
- **d** True
- **e** 2018
- **f** "cake"
- **g** apple

Control structures

2 Give an example of an instruction that is a definition.

3 Give an example of an instruction that is a statement.

4 What is the difference between a definition and a statement?

5 How does a sequence differ from an iteration?

6 What is the difference between a WHILE loop and a DO/WHILE loop?

7 Mary writes some code to iterate over a set of data. It runs exactly once, but no conditions have been met. What types of iterations could Mary have written?

8 State the values that will be returned or displayed in the following examples of pseudocode:

a

```
a ← 4
b ← 2
c ← 3
IF (a > c) OR (b > c) THEN
        RETURN True
ELSE
        RETURN False
ENDIF
```

b

```
age ← 10
IF age < 10 Then
        RETURN "Child"
ELSEIF age < 18 THEN
        RETURN "Teenager"
ELSE
        RETURN "Adult"
ENDIF
```


9780170440943

TEST YOUR KNOWLEDGE

Functions, classes, methods

9 How do you get a function to run within source code?

10 How are functions represented in pseudocode?

11 What is the difference between a function declaration and a function definition?

12 How is a class different to an object?

13 What is the relationship between a method and an event?

14 How are methods represented in pseudocode?

Algorithms for sorting

15 Explain the steps taken to perform a selection sort, using a worked example as part of your explanation.

16 Explain the steps taken to perform a quicksort, using a worked example as part of your explanation.

17 Which is faster, quicksort or selection sort? Is this always the case? Explain.

Algorithms for searching

18 Explain the steps taken to perform a linear search, using a worked example as part of your explanation.

19 Explain the steps taken to perform a binary search, using a worked example as part of your explanation.

20 Roland executes a linear search and then a binary search on a very large set of data. He searches for the same item in each of the searches. The linear search was much faster than the binary search. How is this possible?

Validation techniques

21 Cerie needs to perform all three validation techniques on a particular input. In what order should Cerie perform these checks when she writes her source code? Why?

SCHOOL-ASSESSED TASK TRACKER

☐ Project plan	☐ Justification	☐ Analysis	☐ Folio of alternative designs ideas	☐ Usability tests	☐ Evaluation and assessment	☐ Final submission

9780170440943

22 Annotate the following pseudocode to show where each validation technique is being used.

```
ALGORITHM validateInput()
BEGIN
        INPUT fullName
        INPUT age

        IF fullName = blank THEN
            PRINT "Please enter your full name."
        ENDIF

        IF isInteger(age) = False THEN
            PRINT "You must enter age as a whole number."
        ELSEIF age < 0 THEN
            PRINT "Invalid age."
        ENDIF
END
```

Meeting design specifications

23 What steps can be taken to ensure design specifications are being met?

24 What is the difference between a test case and test data?

25 What is the difference between a syntax error and a runtime error?

26 Dina executes some code on her computer and everything seems to slow down. Eventually, her computer crashes. What has most likely occurred within the code?

27 Why is it important for software companies to release patches?

28 What is the purpose of a truth table in relation to testing?

29 Explain what a boundary value is.

30 Why are trace tables useful when debugging?

9780170440943

APPLY YOUR KNOWLEDGE

1. Write a program that asks a user for their name. Once they have entered their name, the program should say hello to them.
2. Write a simplified calculator program that implements basic arithmetic operations. At a minimum, it should handle addition, subtraction, multiplication and division of at least two numbers.
3. Using the pseudocode in Figures 2.25 and 2.26 on pages 53 and 54, implement linear and binary searches in your selected programming language.
4. Using the pseudocode in Figures 2.20 and 2.24 on pages 46 and 50, implement selection sort and quicksort in your selected programming language.
5. Complete a trace table for the following algorithm:

```
ALGORITHM printResult()
BEGIN
        x ← 2
        y ← 1

        WHILE y < x DO
            y ← x * y
            x ← x + x
        ENDWHILE
        PRINT x, y
END
```

6. Construct a truth table and algebraically determine the test data required to test the following algorithm:

```
ALGORITHM returnResult()
BEGIN
        INPUT x, y, z

        IF x < y AND z > y THEN
            RETURN True
        ELSE
            RETURN False
        ENDIF
END
```

SCHOOL-ASSESSED TASK TRACKER

☐ Project plan | ☐ Justification | ☐ Analysis | ☐ Folio of alternative designs ideas | ☐ Usability tests | ☐ Evaluation and assessment | ☐ Final submission

9780170440943

PREPARING FOR Unit 3

OUTCOME 1

On completion of this unit, the student should be able to interpret teacher-provided requirements and designs, develop and test working modules and justify the use of processing features of a programming language.

Steps to follow

To achieve this outcome, you will draw on key knowledge and key skills outlined in Area of Study 1. This Outcome requires that you use a programming language to create working modules, undertake the problem-solving activities of designing, coding, validating and testing, and create internal documentation in response to teacher-provided requirements and designs. These working modules do not need to represent complete solutions, but the modules themselves should be complete as outlined in the design specifications provided to you.

Your teacher may choose to give you the requirements and designs one at a time after periods of relevant theory and programming instruction, or they may give them to you as a group after all of the theory and programming has been covered. To encourage you to meet all of the requirements, your teacher may choose to allocate different classes to different stages of the task, such as separating development from testing. Your justifications of the use of processing features of a programming language will be included in your internal documentation, alongside descriptions of the modules and their features.

Documents required for assessment

- Source code
- Internal documentation within the source code
- Evidence of testing using appropriate test data
  - Keep and submit all documents used to construct your tests, including test data, any truth tables used to select your test data, and trace tables.

Assessment

You will be assessed on the following measures:

- Your choice of data types and data structures
- Your choice of file types
- Selection, creation and use of appropriate processing features
- Naming conventions
- Validation techniques
- Debugging techniques
- Internal documentation of module functions
- Internal documentation justifying the use of processing features
- Thoroughness of testing

9780170440943

CHAPTER

3 Software analysis

KEY KNOWLEDGE

On completion of this chapter, you will be able to demonstrate knowledge of:

Digital systems

- security considerations influencing the design of solutions, including authentication and data protection
- features of project management using Gantt charts, including the identification sequencing of tasks, time allocation, dependencies, milestones and critical path.

Data and information

- techniques for collecting data to determine needs and requirements, including interviews, observation, reports and surveys.

Approaches to problem solving

- functional and non-functional requirements
- constraints that influence solutions, including economic, legal, social, technical and usability
- factors that determine the scope of solutions
- features and purposes of software requirement specifications
- tools and techniques for depicting the interfaces between solutions, users and networks, including use case diagrams created using UML
- features of context diagrams and data flow diagrams

Reproduced from the VCE Applied Computing Study Design (2020–2023) © VCAA; used with permission.

FOR THE STUDENT

The process of creating documentation related to project management and for the specifications and design of a software system must be systematic in order to ensure success. Throughout both Units 3 and 4 you will be required to maintain a project management progress report in the form of a Gantt chart. The SAT requires that the project management report be submitted twice. In Unit 3 it will be in the form of a proposed plan. In Unit 4 it is an amended plan and, with the benefit of hindsight, includes changes that reflect the actual project progress, rather than the imagined progress as recorded by the proposed plan in Unit 3. Data collection techniques help to determine software requirements, both functional and non-functional, and also help to determine the scope of the system, particularly when considering the constraints faced by the software developer. Data collection must be interpreted using tools such as use case diagrams (UCDs), context diagrams and data flow diagrams (DFDs) to provide an increasingly detailed overview of the system being built. At this stage of the problem-solving methodology, it is also important to determine the criteria that will be used to evaluate the efficiency and effectiveness of design ideas in order to select a solution to develop further.

FOR THE TEACHER

This chapter, along with Chapter 4, covers the theory required for Unit 3, Outcome 2, which is the first part of the SAT. The SAT will be completed in Unit 4, Outcome 1. Chapter 3 focuses on the analysis stage of the problem-solving methodology (PSM). Chapter 4 will deal with the design stage. The focus of this chapter is the core elements of analysis that help students understand the requirements of project management and to construct a software requirements specification (SRS) for the proposed systems. Students should apply data collection techniques to obtain information that will help determine the constraints and scope of a proposed system as well as document its functional and non-functional requirements. As part of this documentation, students must learn to construct use case diagrams using UML, as well as context and data flow diagrams to represent the inputs, processes and outputs of systems and solutions. Students must propose different design ideas for their solution. They must construct and use evaluation criteria to determine the most efficient and effective design.

iStock.com/andrewhoughton

9780170440943

What is a 'software solution'?

Software is the means by which computer hardware can achieve a purpose. The key ingredient of effective and efficient software is that the expectations of the user and the **software developer** are satisfied by the look, feel and function of the software, or program.

What makes a 'good program'?

Over the course of Units 3 and 4, the SAT and PSM will take you through a series of steps. First you will collect information about the needs and requirements that have been identified by 'your client', and then this information or **data** will be transformed into a software requirements specification (SRS). A benefit of identifying the purpose to be achieved by your software is that the **evaluation criteria** will be well known before you get to coding and testing in Unit 4.

As part of Unit 3, Outcome 2 follows a **problem-solving methodology (PSM)** to go through the stages of **analysis** and **design**. Development and evaluation will take place in Unit 4, Area of Study 1.

To prepare you for Unit 3, Outcome 2 and also for Unit 4, Outcome 1, we will discuss planning and managing a complex project, including how to use a Gantt chart, because this relates to the project you are undertaking.

As part of Unit 3, Outcome 2, you first need to find a suitable client who has a need or opportunity for a software solution. The first thing we will discuss is how to determine a client's needs and requirements. A variety of techniques will be considered, including interviews, surveys, reports and observations.

Next, we will talk about the features of functional and non-functional requirements for a solution, both in terms of general solutions and your specific software solution.

Then we will discuss factors such as scope and constraints that will affect the ability to deliver the solution and to satisfy client expectations. Before you begin any software design, you need to create the software requirements specification (SRS). In order to explain your intended software solution, you will use tools and techniques to illustrate the relationships between your solution, users and networks. The tools used will include use case diagrams (UCDs) with a unified modelling language (UML), context diagrams and data flow diagrams (DFDs).

The first thing to do after choosing a client is to gather data about their requirements. You can acquire data through methods such as surveys and reports, and through conducting interviews and observations.

Once you have gathered all of this data, you need to store it, protect it and understand what type of data it is. We discuss how to reference those sources properly and briefly cover data types and structures, which is relevant to Outcome 2, later in Chapter 4.

FIGURE 3.1 Chapter map

9780170440943

Project management

Project management is the process of planning, organising and monitoring a project in order to ensure it is completed on time and within budget and scope. Building or changing information systems for a project can be expensive and disruptive and, if managed badly, can be damaging to an organisation's operations and profit. Large-scale changes are often approached as projects so they can be planned, organised and conducted appropriately, ensuring that they finish on time and within budget, and fulfil the project's goals (scope). You will formulate a project plan to manage your progress through Unit 3, Outcome 2 and Unit 4, Outcome 1.

For your project to be successful, you need to identify, schedule and monitor tasks, resources, people and time. While you can use a software tool for planning a project, our main focus initially will be on the **concepts** and **processes** of project management.

One of the items you are required to submit as part of your Outcome is a **Gantt chart**.

A Gantt chart is a type of bar chart or graphic timeline named after its inventor, Henry Gantt, that:

- lists all tasks in a project
- organises the tasks in order
- shows which tasks must wait for other tasks to finish before they can begin
- allocates people and resources to tasks
- tracks the progress of tasks and the entire project.

Although you can create a Gantt chart with a pen and paper or a spreadsheet, project management software is usually easier and faster. Suitable software includes the commercial Microsoft Project, and the free, easy-to-use GanttProject and ProjectLibre.

When using software to create your Gantt chart, you will not be assessed on your technical prowess with the software. Rather, you will be assessed on how well your Gantt chart demonstrates your understanding of the concepts and processes of project management.

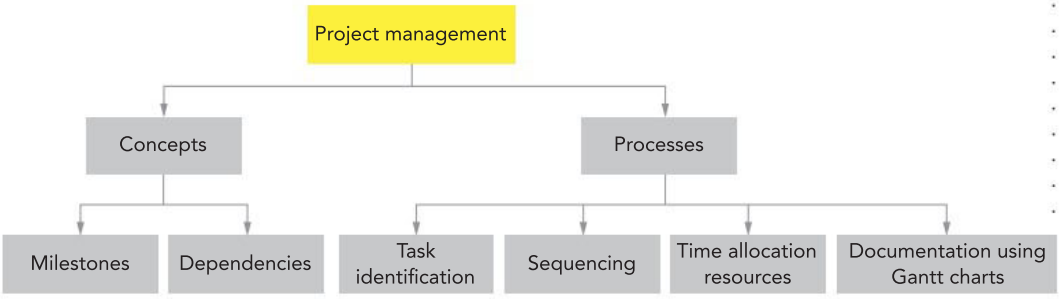


FIGURE 3.2 Key project management concepts and processes

Concepts

Milestones

A milestone represents the achievement of a significant stage in a project and has zero time duration. For example, the completion of the printing of a questionnaire so that it can be distributed to respondents would be a task of zero time and represents a milestone. This follows tasks in which the questionnaire has to be researched, written, proofread and finally printed, all of which take time.

9780170440943

Dependencies

Tasks are interdependent, meaning that they must be completed in a particular order. The commencement of some tasks depends directly on the task that is completed before. For example, you cannot distribute a questionnaire (one task) before writing the questions for it (another task). However, you cannot write the questions for the questionnaire without first determining what information, or data, you need (a third task). Ultimately, the task of distributing a questionnaire has multiple dependencies. It is not possible to distribute the questionnaire without first writing the questions.

Processes

Task identification

You would break a large project such as Unit 3, Outcome 2 down into discrete tasks, such as the following:

- Identify a suitable client.
- Analyse the client's needs and requirements.
- Create a Gantt chart to identify tasks.
- Create a working title for your software solution.
- Write qualitative questions for interviews.
- Conduct interview(s) with the client and possible users.
- Write quantitative questions for questionnaires.
- Distribute questionnaires.
- Locate other data sets and secondary data.
- Collate data.
- Interpret data and create the SRS.
- Modify your SRS after discussion and agreement with your client.
- Finalise the purpose and function of your software solution.

Note: Not all of the tasks you would undertake for Unit 3, Outcome 2 are included in the list above. You should not use this as your own exhaustive list.

To break down your project into achievable tasks, develop a **work breakdown structure (WBS)** and draw a WBS diagram to accompany it. For large projects, a WBS will often be hierarchical, breaking major tasks into subtasks and even sub-subtasks. Although this may sound confusing, it will actually keep your tasks organised and in context, allowing them to be collapsed or expanded to view overall task progress or fine details about how minor subtasks are proceeding.

Do not leave any tasks out of the WBS. For example, imagine you distributed both a print and online version of a questionnaire, but did not list the printed version in the WBS, and forgot to collect the printed forms from respondents. All the gathered data would be overlooked, or counted too late.

Figure 3.3 shows a sample potential WBS for Unit 3, Outcome 2. Again, the tasks may not be exhaustive and you may find that your own WBS requires additional tasks.

9780170440943

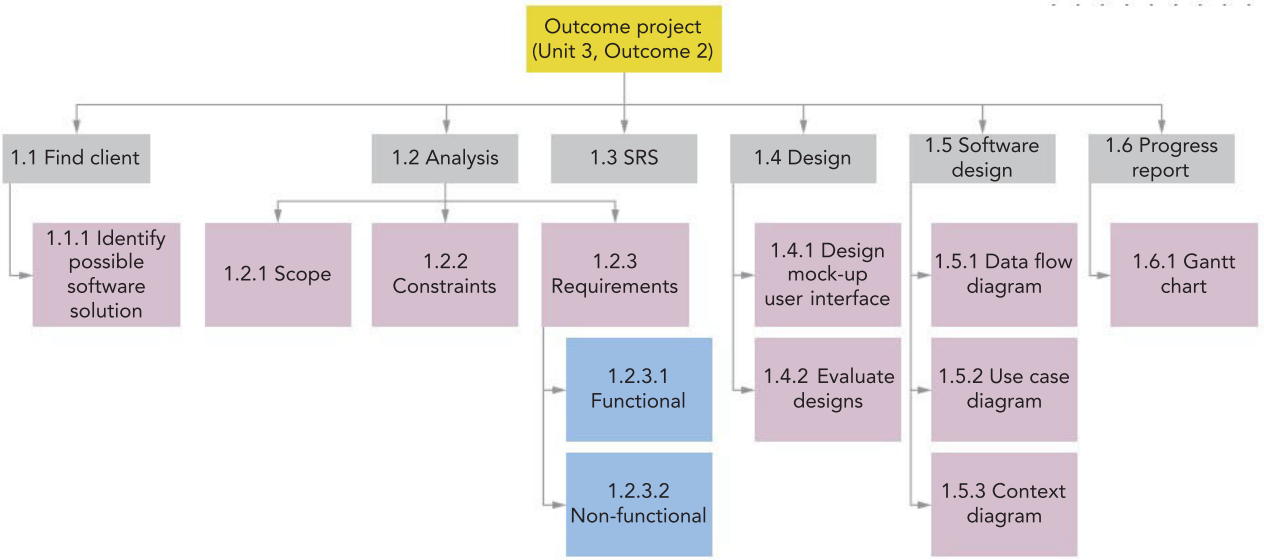


FIGURE 3.3 Sample breakdown structure (WBS) diagram for Unit 3, Outcome 2

Sequencing

When you have identified each individual task, you need to estimate how long each task will take and then put them in a sequence; that is, arrange them in a particular order. As discussed in dependencies, one task often cannot be started before one or more other tasks have been completed.

Decide which tasks can be worked on **concurrently**, but are dependent on other tasks that have already been completed. For example, you could work concurrently on conducting interviews, researching similar applications and writing questions for a survey, but these tasks are all dependent on having a client with a need or opportunity.

Similarly, you cannot write an SRS before you have interpreted the data that you have collected in the analysis stage.

Tasks that must be completed before another task can begin are called **predecessors**. The dependent tasks are called **successors**.

If a predecessor runs overtime, all of its successors will be delayed, causing problems for other tasks and deadlines. This is where a Gantt chart becomes very useful – it helps to monitor tasks and meet deadlines, keeping the project on schedule. It also helps you to visualise the problems that will occur down the line if a predecessor is late.

The amount of time that a task can be delayed without delaying another task, or the project completion date, is called slack time. When workers have **slack time**, you can reassign them to other tasks.

Time allocation resources

A Gantt chart shows tasks as horizontal bars. Each horizontal bar is of a length proportional to the task's duration. A very short task will have a very short bar, while a very long task will have a very long bar. Figure 3.4 displays a number of features typical to a Gantt chart.

The names of the tasks are shown in the left pane, along with start and end dates, while the right pane shows task timelines. Tasks that overlap in time are concurrent and can be carried out at the same time using different teams.

9780170440943

Arrows are used to indicate dependency. For example, neither the 'Functional' nor the 'Non-Functional' tasks under 'Requirements' can begin until all tasks under 'Analysis' have finished, because they depend on these tasks.

The diamond shape indicates a milestone. Milestones are points of significant progress in a project. They are often the start or end of major stages, and can be used to monitor whether a project is on track. A milestone is an **event** with zero duration and no allocated resources. It is simply shown as a diamond-shaped 'task'. In this instance, 'Submit final report', the end of the project, is one such milestone.

An event differs from a task because although something happens (for example, a major task ends), no resources, work or time are allocated to it because there is nothing that people need to do to make it happen.

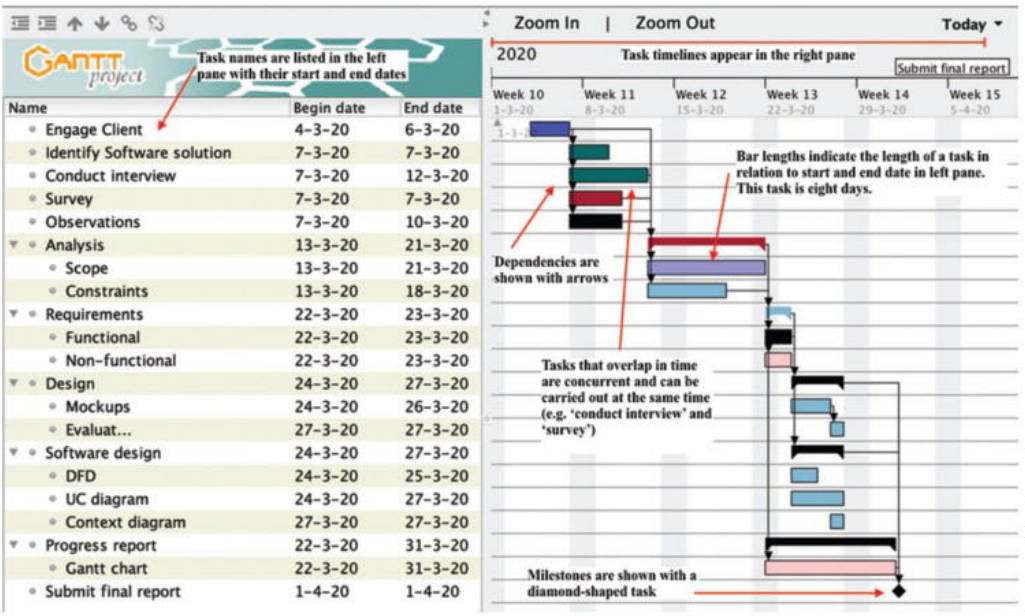
A project's **critical path** is the sequence of tasks from beginning to end, which:

- contains no slack time – any delay in a task on the critical path will affect the end date of the project
- is the longest duration
- is the minimum possible time in which all of the project's tasks can be completed.

Sometimes, more than one critical path is possible. No task on the critical path can have its duration changed without affecting the end date of the whole project.

Documentation using Gantt charts

You can use the Gantt chart you develop to mark your progress throughout this Area of Study. Including information about the progress of the task and the planned versus actual duration of the task will help keep you on track.



Name	Begin date	End date
Engage Client	4-3-20	6-3-20
Identify Software solution	7-3-20	7-3-20
Conduct interview	7-3-20	12-3-20
Survey	7-3-20	7-3-20
Observations	7-3-20	10-3-20
Analysis	13-3-20	21-3-20
Scope	13-3-20	21-3-20
Constraints	13-3-20	18-3-20
Requirements	22-3-20	23-3-20
Functional	22-3-20	23-3-20
Non-functional	22-3-20	23-3-20
Design	24-3-20	27-3-20
Mockups	24-3-20	26-3-20
Evaluat...	27-3-20	27-3-20
Software design	24-3-20	27-3-20
DFD	24-3-20	25-3-20
UC diagram	24-3-20	27-3-20
Context diagram	27-3-20	27-3-20
Progress report	22-3-20	31-3-20
Gantt chart	22-3-20	31-3-20
Submit final report	1-4-20	1-4-20

Courtesy of GanttProject

FIGURE 3.4 An annotated Gantt chart. This Gantt chart provides a partial sample model for Unit 3, Outcome 2, with placeholder dates.

NelsonNet additional resource: Figure 3.4 Sample GanttProject chart

9780170440943

To manage your solution effectively as a complex project, you should also use your Gantt chart to document the resources you have allocated to it, such as any tools and equipment. While a company or organisation may list consultants and buildings as resources, your resources might be computers, particular data sets and software tools.

You should also frequently modify your Gantt chart to reflect contingencies. A contingency is an unforeseen event, incident or emergency. You may find that your client suddenly becomes unavailable or unwilling to continue. The software language you wish to use is no longer acceptable. The website hosting your analysis and results crashes and you lose your data. Your Gantt chart should show problems like these and how you react to them, such as finding a new client or a different software solution strategy, or switching languages.

You should keep your Gantt chart updated throughout both Unit 3, Outcome 2 and Unit 4, Outcome 1. You will submit an initial project plan, indicating times, resources and tasks, in Unit 3, Outcome 2. After modifying the plan to indicate changes, you will submit an evaluation of the plan in Unit 4, Outcome 1.

THINK ABOUT SOFTWARE DEVELOPMENT 3.1

Project management tools are useful to find the perfect number of people needed on a task so it is finished as quickly as possible without anyone being idle. Using software, develop a Gantt chart to plan the baking of a cake. Assume you can use as many cooks as you want.

CASE STUDY

Gantt chart for creating a database

Several web developers are working on a database project together. As part of their project, they need to build a Gantt chart.

Task identification

They first identify the tasks they need to complete using a WBS diagram (see Figure 3.5). Next, they enter these tasks into their chosen Gantt software, GanttProject (see Figure 3.6).

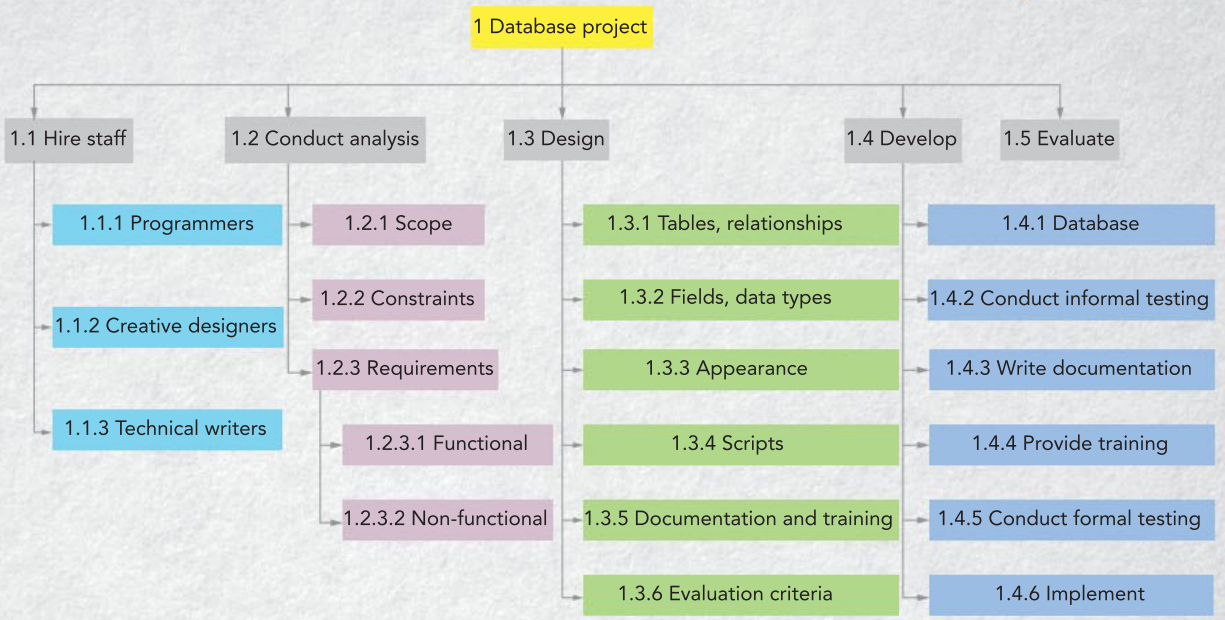


FIGURE 3.5 Task identification

SCHOOL-ASSESSED TASK TRACKER

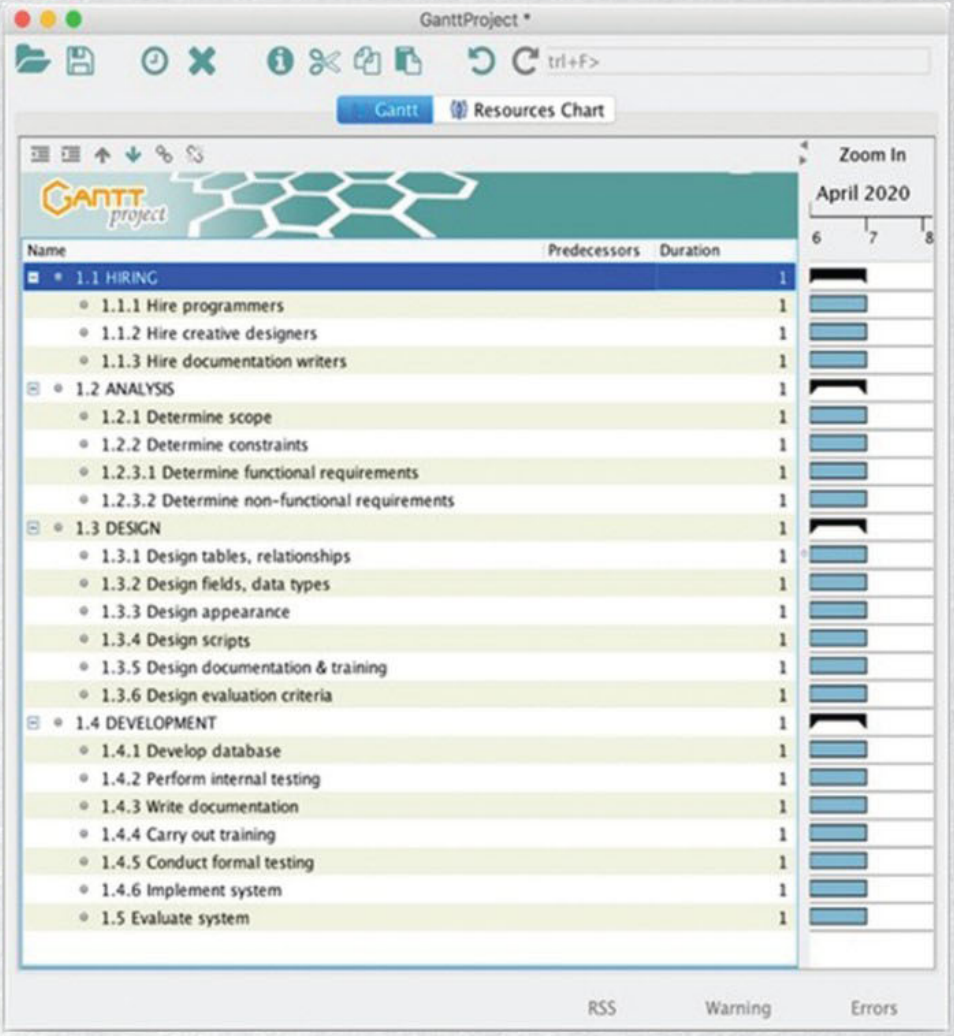
☑ Project plan	☐ Justification	☐ Analysis	☐ Folio of alternative designs ideas	☐ Usability tests	☐ Evaluation and assessment	☐ Final submission

9780170440943

THINK ABOUT SOFTWARE DEVELOPMENT 3.2

In terms of project management, research the meaning of:

- an 'optimistic' task duration
- a 'pessimistic' task duration.



Courtesy of GanttProject

FIGURE 3.6 Entering tasks into Gantt software

The developers use a hierarchical structure to group tasks under headings, such as HIRING and ANALYSIS, to make task management easier. Groups of tasks can be collapsed or expanded or moved as a group. In GanttProject, multiple levels of sub-tasks can easily be created by just indenting them in the task properties.

Sequencing

The tasks are sufficiently sequenced, but the order can be shifted easily if needed. The developers start creating dependencies, forcing dependent tasks to wait until their predecessors have finished.

- All of the HIRING (1.1) tasks can start immediately.
- Management will complete the tasks in the ANALYSIS group (1.2), which can also begin immediately and run concurrently with the hiring tasks.
- The DESIGN (1.3) tasks cannot begin until the ANALYSIS tasks are complete, so DESIGN is made dependent on ANALYSIS.
- The DEVELOPMENT (1.4) tasks cannot begin until DESIGN is finished, so DESIGN is as a predecessor to DEVELOPMENT.
- EVALUATE SYSTEM (1.5) must wait for everything else to finish, so it is made dependent on DEVELOPMENT.

9780170440943

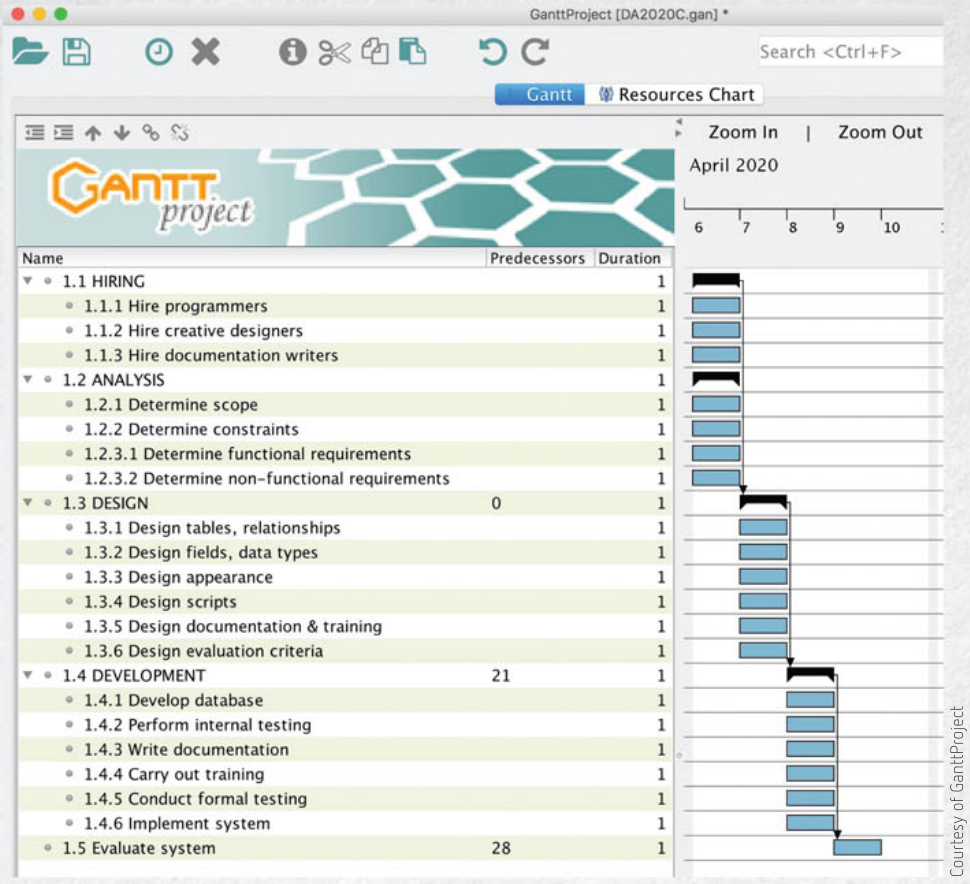


FIGURE 3.7 Major dependencies have been added. Arrows lead from predecessor tasks to dependent tasks.

Subtler dependencies can now be added.

- The team wants DESIGN to be well underway before they start creating documentation and training, because they would be greatly affected by changes to the database. They make 1.3.5 dependent on other design tasks being finished.
- They also make 1.4.4 (Carry out training) dependent on 1.4.3 (Write documentation) being finished.
- Formal testing (1.4.5) must follow all database creation tasks, so they add another dependency.
- Implementation (1.4.6) comes as the last stage of development.

They are now happy with the logical task sequences and dependencies, but they know that if their needs change later, they can easily adapt the chart to suit their needs.

Time allocation resources

The project developers now tackle the challenge of the time required for each task. They consult with experts and colleagues, and use their extensive experience and knowledge of past projects to guide their estimates.

The developers can finish with their Gantt chart by:

- selecting a project start date
- showing a critical path (critical tasks are shown cross-hatched)
- adding milestones to mark the end of major stages of the project.

9780170440943

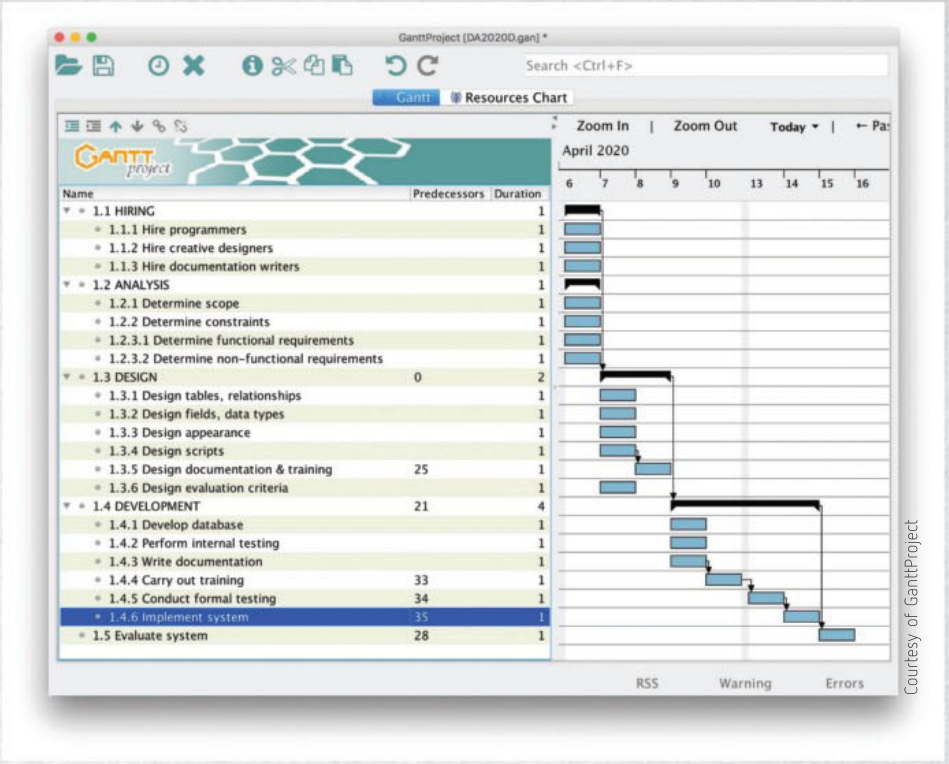


FIGURE 3.8 Dependencies have been created

Additional resources

NelsonNet additional resource: Figure 3.8 Gantt chart showing task durations

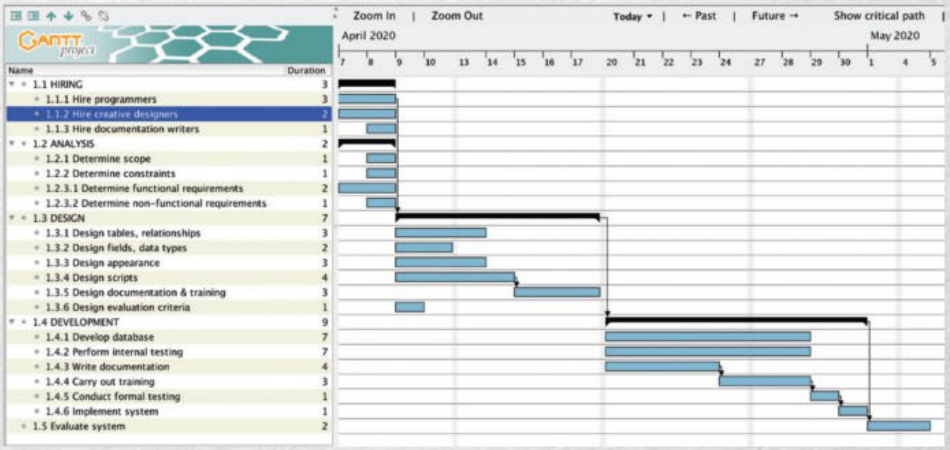


FIGURE 3.9 Task durations have been estimated

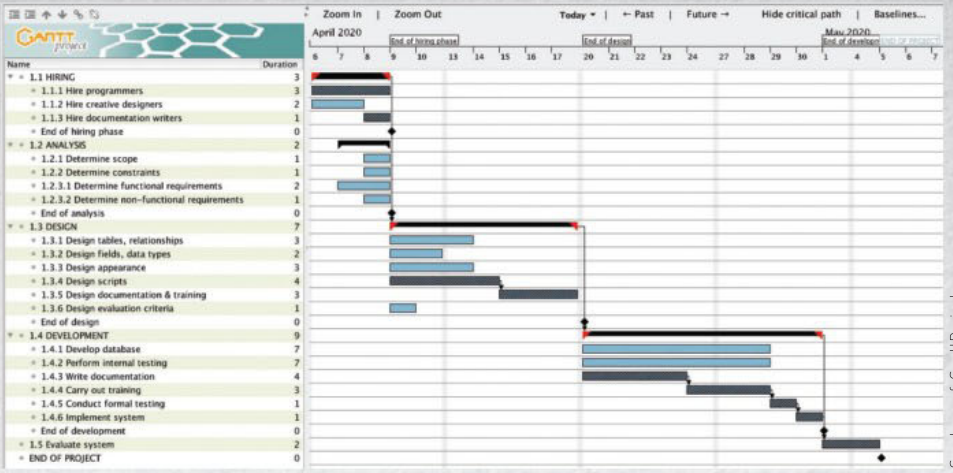
Additional resources

NelsonNet additional resource: Figure 3.9 Gantt chart finished first draft

Questions

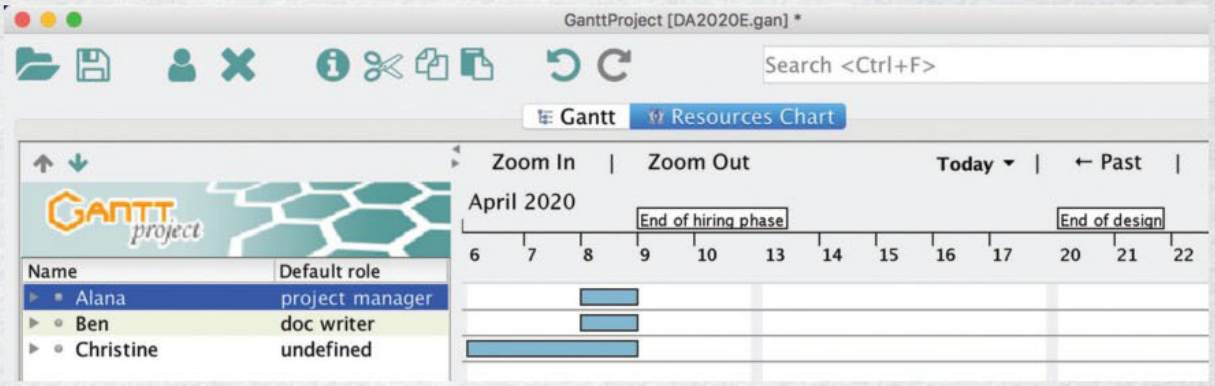
1. The project team has made a mistake with the starting date of the 'Evaluate system' task. Explain why. How could they fix it?
2. How much slack does the 'Hire creative designers' task have?
3. If designing scripts took a day longer than expected, would it affect the project end date?
4. The team discovers that the 'Develop database' task is running over time. How could they keep the project from running past its planned end date?
5. Explain the benefits of using Gantt chart software instead of using a pen and paper or a whiteboard.

9780170440943



Courtesy of GanttProject

FIGURE 3.10 Finished first draft of the Gantt chart



Courtesy of GanttProject

FIGURE 3.11 Adding resources to the project

Activity

6 Obtain a Gantt software tool and create a simple chart of your own.

Documentation using Gantt charts

To ensure that project workers are not booked to be in two places at once, or idle, and that equipment is ready at the right time, the project manager allocates resources to tasks using the Gantt chart.

Once the project is underway, the team will continue to refer to the Gantt chart to monitor their progress, and they will modify the chart when contingencies force plans to change.

While Gantt charts are one crucial aspect of project management, good file management practices are another. Wise file-naming strategies are easy to learn and useful in many ways. You will find it easier to keep track of the materials you collect for your solution if you learn to manage your files by naming them wisely.

SCHOOL-ASSESSED TASK TRACKER

☑ Project plan	☐ Justification	☐ Analysis	☐ Folio of alternative designs ideas	☐ Usability tests	☐ Evaluation and assessment	☐ Final submission

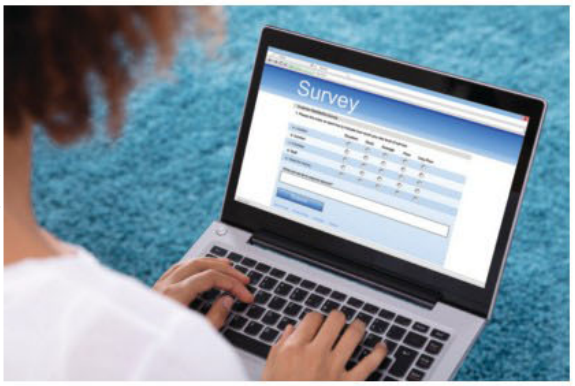
9780170440943

Collecting data

Data to construct a software requirements specification (SRS) is usually collected through methods such as surveys, interviews, reports and observations. Each method of data collection has its strengths and weaknesses, and it is important that the most relevant methods of data collection are used within a given context. In Software Development, data that is collected typically helps determine the project scope, the functional and non-functional requirements, and the constraints. It can also help determine and/or decide who the end users will be, or how the client is going to use the software.

Survey

A **survey** is usually a set of questions that ask for a response to be selected from a list of alternatives, such as A, B, C, D; or a range, such as 1–5 or very low to very high. This type of survey is also called a questionnaire. Surveys can easily be given to many people, and are quickly processed and analysed using computer-based methods because the answers can be recorded as numbers as long as **close-ended questions** are used. Close-ended questions are questions where the answers are either boolean (yes or no) or ranked on a fixed scale. These types of questions allow for analysis of **quantitative data** to produce results, which is more efficient than surveys with **open-ended questions**. Open-ended questions are questions that ask for answers in sentences or worded form. This means the number of answers is potentially infinite. These questions tend to ask for opinions, and must undergo analysis of **qualitative data** to interpret the results.



Shutterstock.com/Andrey Popov

FIGURE 3.12 A survey with a close-ended question

The advantage of administering a survey is that they are relatively inexpensive to conduct. They can also be delivered digitally, so results can be collected immediately, and the survey can easily be given to a large number of users to complete. The disadvantages are that processing of results can take a lot of time if open-ended questions are asked, users may not necessarily be truthful in their responses and, if the survey is quite lengthy, users may lose interest in completing it.

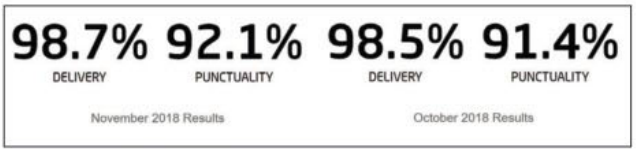
Interview

An **interview** is usually conducted face-to-face, sometimes in groups, and can take a substantial amount of time. Interviews give an opportunity for in-depth follow-up and the ability to ask clarifying questions – this cannot be done with a survey, which is often answered in private. Interviews are very useful for eliciting feelings, attitudes and opinions that are too complex to easily record in a survey.

Report

A **report** is typically a written document providing a summary of findings in relation to the system being analysed. Often when a software developer is creating a new system to replace the system that is currently in place, they must investigate the current system as part of their analysis of the requirements of the new system. In this instance, it can be useful to collect

9780170440943



Metro Trains

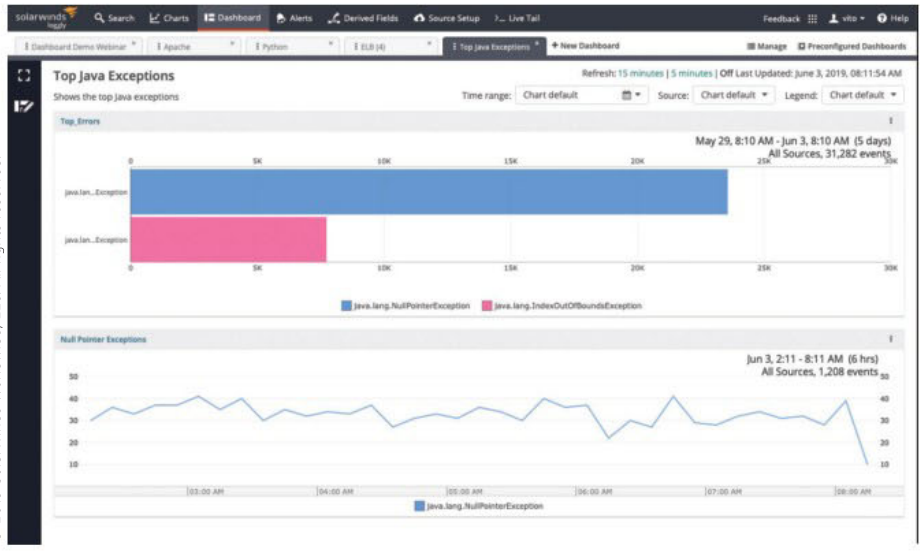
FIGURE 3.13 Metro train performance report

data about the current system; often this data has been summarised in report documents. These could include error reports, customer complaint summaries, uptime reports and other related system reports such as system performance reports and monitoring reports. The types of reports collected will vary depending on the system being built and the system being replaced.

The advantages of using reports as part of data collection is that they are often pre-prepared and quick to obtain, and therefore inexpensive. The data within them has already been collated and interpreted, saving time (Figure 3.13). This can also be a disadvantage, however, as the method of interpretation may not be sound, or may have been deliberately manipulated to present a particular point of view. Using data from reports can be risky if the source of those reports is not reliable (Figure 3.14).

www.dotcom-monitor.com

FIGURE 3.14 Server uptime report



SolarWinds® Loggly® top Java exceptions.
© 2019 SolarWinds Worldwide, LLC. All rights reserved.

FIGURE 3.15 Error log report

SCHOOL-ASSESSED TASK TRACKER

☑ Project plan	☑ Justification	☐ Analysis	☐ Folio of alternative designs ideas	☐ Usability tests	☐ Evaluation and assessment	☐ Final submission

9780170440943

Observation

Observation is a method of data collection that involves physically observing how a system operates and how it is used. Observations are typically performed when designing a system that will replace a current system, and are useful when attempting to solve a problem with an existing system. For example, a system may become particularly slow at certain times of the day, but will work well outside of those times. An observation of users at the time when the system slows would be useful in helping determine why the issue has occurred. Similarly, systems may have wildly varying performance depending on the time of year, such as consumer websites during periods of high sales such as Boxing Day or the end of the financial year (EOFY). Observations can also be used when creating a new system that will replace something that is currently being completed manually. In these instances, the manual process is observed and documented.

Advantages of observations are that they can provide an unbiased view of the system; information can be gathered without asking for a user's opinion or relying on their memory. Observations can also occur simultaneously as long as users are located in the same physical space. Disadvantages are that they are quite time-consuming and therefore very expensive if they need to occur more than once or over a long period of time. Time-specific observations can also be very difficult if the project timeline does not allow for it; for example, waiting for the end of the financial year may require waiting 12 months. Similarly, if the observations occur at the wrong time, the results may not be very useful as they may not be representative of how the system is typically used. Another disadvantage is that people can become very self-conscious when they are being observed, so their actions may not necessarily be the same as if they were alone, resulting in inaccurate data.

Functional and non-functional requirements

Solution requirements are what the client needs from a solution; that is, what the system must do. These can be broken down into functional requirements and non-functional requirements.

Functional requirements

Functional requirements are directly related to what the solution will do. These typically involve calculations; data processing; opening, reading and writing to files; data manipulation such as image editing; and other specific **functionality** required within the system.

Some examples of functional requirements for software solutions are:

- save customer data to a file
- calculate discount values on products
- set an alarm to go off at a particular time
- load a set of jobs into a timetable.

Functional requirements are usually described in terms of the inputs required, the sequence of operations that will be performed, and the output(s) after processing has occurred. These written descriptions often include dot-point descriptions for the purposes of clarity. An example of a functional requirement for a music performance system can be seen in Figure 3.16.

Functional requirements that require user interaction are typically accompanied by a **use case**; these are described in more detail later in this chapter.

9780170440943

3.1.1. Add Performance

This screen will allow the entering of performance information. Information consists of the performance name, date and theme, as well as a unique performance ID (numeric). Also recorded is whether the performance is a concert, and, if so, the theme of that concert.

Inputs:

- Performance ID (unique)
- Performance Name, Performance Date
- Whether it is a concert or not
- Concert Theme

Sequence of Operations:

- Input will come in from the User Interface, all items except concert theme are required fields
- Themes should be retrieved from a file containing all available themes so a user can select one
- No two concerts should have the same ID
- A theme only needs to be selected if it is a concert
- No concert should be added if a concert with the same theme has occurred in the last six years
- Once data is entered and validated, it should be saved to a performance file

Outputs:

A pop-up box should be displayed showing the success or failure of saving the concert information. Once information is saved, the user should be returned to the main menu.

FIGURE 3.16 A functional requirement for a music performance system

Non-functional requirements

Non-functional requirements are other requirements that the user or client would like the solution to have but that do not affect what the solution does. These tend to be referred to as *quality requirements*, as they typically involve criteria that can be used to 'judge' a system, rather than criteria that involves specific behaviour the system is required to have. Non-functional requirements are often tied to the **constraints** of the system. They can be categorised in terms of usability, reliability, portability, robustness and maintainability.

Non-functional requirements must be measurable – this means that they must be able to be tested to see if the requirement is met.

Determining non-functional requirements usually involves discussions with a client, such as asking if the software must work on different operating systems, or asking who the users of the system will be and the level of technical experience they have.

Usability

Usability relates to how easy a system is to learn and use. This is typically described in terms of efficiency and effectiveness. Common factors of usability include the **clarity** of the user interface and the intuitiveness of the functions within the system. The success of a system's usability is often measured in terms of user satisfaction. An example of a non-functional requirement related to usability is 'users should be able to use all basic functions after one hour of training'.

9780170440943

This non-functional requirement is highly reliant on the skills, expertise and needs of the intended users of the system.

Reliability

The **reliability** of a software solution relates to how much it can be depended upon to function as designed, and for how long. Typically, this requires that the software is deemed **fit for purpose** over time and that it is resistant to failure.

The reliability of a system is generally expressed as a probability measure, where *reliability = 1 – the probability of failure*. An example of a reliability measure is the prediction of the **uptime** of a system or solution, such as 'the system should have a 99.9% uptime over any 12-month period' or 'the ability to add a new customer should be available to staff members during working hours on weekdays'.

Portability

The **portability** of a software solution relates to how easily it can be used in different operating environments. This can involve measuring how easy it is to move from one system to another system with the same architecture; how easy it is to reinstall a program on a new system; and the ability to use the same software on multiple operating systems and platforms, or in multiple languages.

The most time-consuming element of portability is when a developer is required to write software that will work on multiple operating systems and platforms, in particular when multiple languages are involved. This typically means that the user interface must be separated from the core functionality and logic so it is easy to create a new user interface for a new system; for example, creating an application that works on a mobile device as well as a desktop computer, in Mandarin as well as in English.

Portability is increasingly important as mobile computing becomes the norm; users often expect that applications will work on mobile technology as well as desktop technology.

Robustness

The **robustness** of a system relates to how well a software solution responds to errors that occur when the software is being used; that is, it should perform correctly in every situation encountered by a user. Robustness is therefore an evaluation of the error handling techniques within a software solution. For example, a piece of software would be evaluated according to how it responds to bad input from a user. Robustness is closely linked to the use of validation techniques, as these can help prevent errors from occurring when users enter unexpected or invalid input. An example of a non-functional requirement related to robustness is 'the system should reject invalid data entered by a user'.

Robustness is measured in terms of the number of failures, crashes and errors that occur while a system is running.

Maintainability

Maintainability is related to how easy it is to look after software once it is being used. This can involve fixing errors in the code, maximising efficiency and reliability, installing the software on new systems, and in some cases expanding on the current functionality with new functionality. Simply put, maintainability is measured in terms of how easy it is to fix, modify or change the software once people are using it.

Often, maintainability is measured in terms of the number of hours a developer or administrator spends to keep the system running after it has been put in place. An example

9780170440943

of a non-functional requirement related to maintainability is 'fewer than 10 hours should be spent per quarter on maintaining the system'.

Scope

The scope of the software solution must be defined. A project without limits will never be finished, and inevitably disputes will arise over what was and was not included in the expected software solution. A scope statement identifies who will develop the solution and what will be specifically considered, and it may specifically exclude other areas. In its simplest terms, the scope of the project describes the project and explains what the project will and will not do (see pages 13–14).

Constraints

The **constraints** on a software solution are anything that may limit the software developers' options for development or delivery. The software requirements specification (SRS) should contain any decisions made before the project begins. Examples include: time and budget available, programming languages, software processes, hardware limitations, operating systems, specific design tools, previously purchased components or use of licences, safety and security considerations, legal compliance requirements and privacy laws. The constraints statement could also include any assumption or dependencies that will affect the developers' ability to provide the software solution. For example: A new power supply will be required to power the servers, and a cooling system for the server room. This is not part of the software solution, but successful implementation will rely on the change to infrastructure.

Software requirements specifications

A **software requirements specification (SRS)** is a single document that contains the outcomes of the analysis stage of the problem-solving methodology. This document is created after data collection has occurred and before the **design stage** begins.

An SRS must outline all of the elements considered in the analysis stage. In particular, the *constraints* under which the system must exist, the *scope* of the proposed system (what it will and will not contain) and the *functional* and *non-functional* requirements of the system itself. An SRS may also include an appendix containing additional information needed to interpret the requirements, such as a description of the operating environment of the proposed system (linked to constraints) or descriptions of any third-party tools required (linked to functional or non-functional requirements).

A well-written SRS provides quality assurance for the client that the issue to be solved by the software solution is well documented and understood. It ensures that:

- the client's problem or opportunity is understood, the issues have been identified, and in response a systematic process of addressing each issue has been documented
- the completed SRS will be the basis for the design specifications. The criteria identified in the SRS will be used to evaluate the success of the final product.
- the final evaluation will verify the software product and test that the software performs as expected.

9780170440943

Shutterstock.com/Bakhtiar Zein

FIGURE 3.17 Software requirements specification

Scope creep is when a client changes the scope of a project (by increasing it) during the life cycle of the project. This can be very expensive to a developer in terms of time and money, as it means the project will take longer to complete, often without any extra money being provided by the client.

An SRS can contain performance parameters such as response time, processing time, maintenance requirements, resource demands, and security and backup arrangements.

Diagrammatical tools that can be used to assist in the creation of the SRS include use cases, context diagrams and data flow diagrams.

An SRS is useful because it provides all of the required information about the proposed system in one place. It often also acts as a legally binding contract between a client and a developer, preventing a client from requesting additional features as the project timeline progresses.

Creating a software requirements specification

An SRS should include a cover page that states the name of the project, the author (or authors), the contact details of the author(s) and the version of the proposed system.

The first page after the cover page should contain a table of contents that clearly lists each section of the SRS. Each section inside the SRS should use numbered headings and subheadings so each element of the document can be easily tracked. Page numbers are also included on each page.

Any additional documentation should be included at the end of the SRS as part of a set of appendices. This may include reports collected in the analysis stage, evidence of interviews and observations, etc.

Because an SRS is a professional document, consistency in the use of fonts, font sizes and font colours is also important.

Interfaces between solutions, users and networks

There are many useful tools to interpret data collected in the analysis stage of the problem-solving methodology. Three methods of depicting the interfaces between solutions, users and networks are use case diagrams (UCDs), context diagrams and data flow diagrams (DFDs).

Use case diagrams

A **use case diagram (UCD)** is a method of describing how a user interacts with a system. This is created using a **Unified Modelling Language (UML)**, which is a general-purpose visual modelling language. Use cases are a diagrammatical representation of the externally visible user interactions, and are often used to complement worded descriptions of those

interactions. Use case diagrams are normally completed prior to the creation of context diagrams or data flow diagrams.

A use case diagram provides a structured view of the functionality of a software solution; it 'tells a story' of how the functions within a system work. It is intended to provide a high-level view of how a user actively interacts with the system, and does not show the internal functionality of the system itself. Use case diagrams are not designed to show the sequence or order of interactions a user undertakes within a system, either.

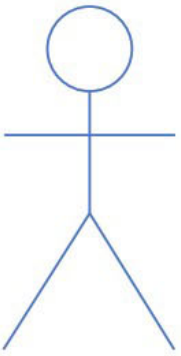
Use case diagrams include actors who have relationships with use cases.

A use case diagram should show non-technical people what the system will do when it is completed. It should show developers what is expected of an application. It does not go into detail about *how* a system will implement functionality; it only shows *what* it will do.

Actor

In a use case diagram, an **actor** represents an entity that can interact with the software solution. While this typically means a human user, it does not have to be, and can include external systems. Actors are described in terms of the role of a user or external system, rather than as specific users themselves. For example, if Serai was a manager in charge of adding users in a new system, she would be represented by an actor named 'manager' rather than by an actor named 'Serai'.

Actors are represented by stick figures in use case diagrams, even if they are not humans. They can be connected to use cases and other actors through relationships and generalisations, and should always be described using nouns.



Actor

FIGURE 3.18 An actor in a use case

Use case

A use case diagram contains **use cases**, which describe transactions or functions a user (actor) can complete in the system. These represent system functionality. Use cases are shown using an ellipse with the name of the function written inside. Each use case is often quite broad, as a use case diagram is often the first tool used to understand how a system will work. They are generally expanded upon in other diagrams, such as context diagrams and data flow diagrams, or in the functional requirements.

Use cases should begin with strong verbs that describe the action or function being represented in the use case. The phrase inside the use case should be brief, often no more than two or three words. For example, 'add customer' or 'delete user'.

Use cases can be connected to actors and other use cases through relationships and generalisations.

FIGURE 3.19 A use case in a use case diagram

Relationship

A **relationship** in a use case diagram represents the connections between elements within the use case diagram. Relationships can exist between actor and use case, actor and actor, and use case and use case.

Association

A typical relationship is represented by a solid line connecting two elements. This type of relationship is referred to as an **association**. This is the default relationship within a use case diagram.

Generalisation

A second type of relationship is a **generalisation**, which indicates a type of inheritance, or parent–child relationship between the two elements. The 'child' in a generalisation gains all of the structure, behaviour and relationships that the parent has. A parent element can have

9780170440943

more than one child element. This feature of a use case diagram is often used to show when the same job can be performed differently by different users. It can also be used to show different types of use cases.

Generalisations are represented by a solid, straight line with a closed arrow head, with the child use case pointing to the parent use case. An example of a generalisation is shown in Figure 3.20, where two actors are connected via a generalisation. The 'Administrator' actor in this example can also perform the role of the 'User' actor. Another example is shown in Figure 3.21, where a parent use case 'process payment' has child use case 'process paypal payment' connected via generalisations.

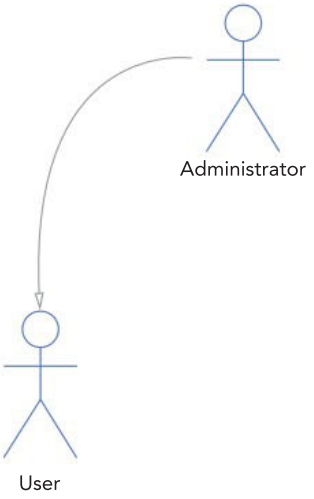


FIGURE 3.20 A generalisation of actors where an administrator is a type of user

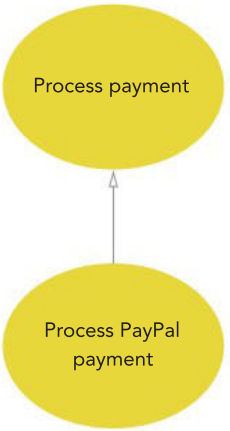


FIGURE 3.21 A generalisation of use cases

Include and extend

Two other types of relationships are include and extend. These are shown through the use of dotted or dashed lines with an open arrow head connecting two use cases; they are not used to connect actors. These relationships must be labelled to indicate whether they are an inclusion or an extension.

When **include** is used, it indicates that the entirety of one use case is included in another. This means that the process of completing a use case always requires running the functions in the other use case at least once. For example, in Figure 3.22, the use case shows that 'edit user' includes 'load user'. This is because to edit a user, that user must first be loaded. Similarly, 'update user details' also includes 'load user', as that functionality also requires that a user be loaded first. Notice that the arrow head on the *include* relationship is pointing towards the use case that is included in the other use case. The *include* relationship must always be labelled using two angled brackets on each side of the word 'include', as shown in Figure 3.22.

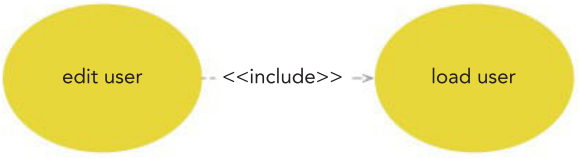


FIGURE 3.22 A use case using 'include'.

When **extend** is used, this indicates that one use case can sometimes be included in another. It is often used to represent additional or optional functionality within the system. For example, in Figure 3.23, the 'move image' use case is an extension of the 'select image'

use case, as moving the image is an action a user can choose to complete after selecting an image. Similarly, in Figure 3.24, the 'display help' use case is an extension of the 'register user' use case, meaning that users can choose to display help relevant to registration, but do not necessarily need to in order to complete the registration process. Notice that, in both examples, the arrow head on the extend relationship is pointing towards the use case that is being extended upon. A use case should not be an extension of another use case if it is required as part of the functionality of that use case; in these cases, the relationship should be an 'include'. The *extend* relationship must always be labelled using two angled brackets on each side of the word 'extend', as shown in Figures 3.23 and 3.24.

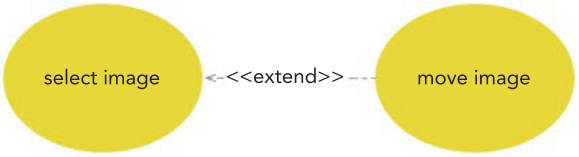


FIGURE 3.23 A use case for an image system using 'extend'.

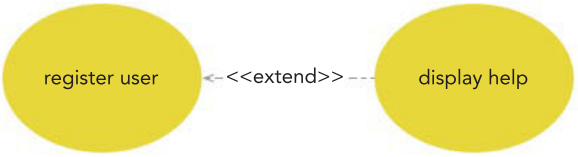


FIGURE 3.24 A use case for a registration system using 'extend'.

System boundary

Use case diagrams show the use cases contained with a **system boundary**. This makes it clear what is included in the system being built and what is not. System boundaries are shown by drawing a rectangle around the use cases that are relevant to the proposed system. Actors should remain outside of the system boundary box. Although there is typically only one system boundary, in cases where scope has been reduced, system boundary boxes can be nested to indicate which use cases are within the current scope of the project, and which have been deemed out of scope. This shows which use cases are part of the scope of an SRS and which may be included in later versions of the software.

Drawing use case diagrams

For consistency, use cases are usually ordered so that all of the main use cases are in the centre of the diagram, typically running vertically, from top to bottom. The order of the use cases does not normally matter, but it can be helpful if similar actions are grouped together. Use cases that are extensions or inclusions of other use cases are typically placed to the right of the use cases they have a relationship with; this often creates two vertical rows of use cases, as shown in Figure 3.25.

Actors are shown on each side of the use cases, and can be repeated (i.e. one on each side) to avoid overlapping relationships; this adds clarity and makes the diagram more readable. System boundaries must be labelled,

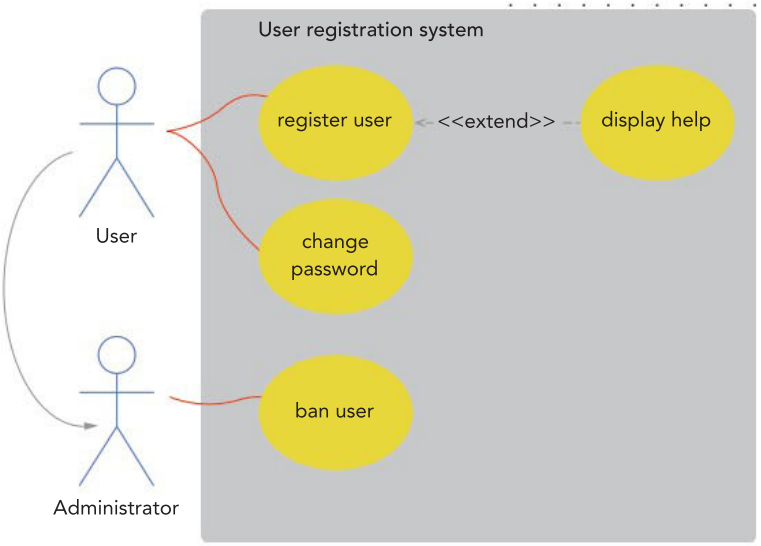


FIGURE 3.25 A use case diagram for a user registration system

9780170440943

typically in the top left corner or bottom right corner of the boundary. Use cases should be labelled within the ellipse, and actors should have labels underneath the stick figure.

Context diagrams

A **context diagram**, sometimes referred to as a Level 0 data flow diagram, is a visualisation of a system in its entirety that indicates the data that is passed into and out of the system. Context diagrams are often created after use case diagrams have been drawn, as all of the interactions a user can have with the system have been documented in the use cases. This allows all of the actions a user can complete to be systematically translated to the data that is needed for those actions to succeed.

Context diagrams do not typically show much detail. They are only intended to focus on the flow of data in and out of the proposed system. This helps establish the context and the boundaries of the software system being created.

There are three primary symbols used in context diagrams, which represent processes, entities and data flows.

FIGURE 3.26 A context diagram process

Process

A **process** in a context diagram is an abstract representation of the whole system being created. Unless the proposed system is very complex, a context diagram will typically contain only a single process. This is shown on the diagram using a circle, with the name of the system contained within that circle.

In a context diagram, the system process is connected to external entities by the use of data flows.

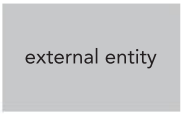


FIGURE 3.27 A context diagram entity

Entity

A context diagram may contain one or more **entities**, which are the users or external systems that interact with the system being created. These are drawn as rectangles within the context diagram, with the name of the entity contained within the rectangle. Much like actors in use case diagrams, entities should not be labelled with the names of real people, but instead with the abstract role those people have.

In a context diagram, entities can only interact with the system process and cannot interact with each other.

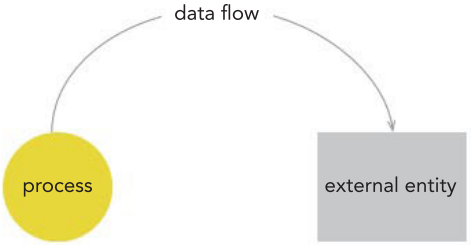


FIGURE 3.28 A context diagram with a process, entity and data flow

Data flow

A **data flow** represents a single piece or logical collection of data as it moves into and out of the system being represented in the context diagram. These are represented by solid lines, typically curved, with an open arrow head representing the direction of the data flow. While most data flows are unidirectional, there are instances where data flows in both directions. In these instances, both ends of the data flow should show open arrow heads.

In a context diagram, data flows connect entities to the system process. They must always start or end at a process, and cannot directly connect entities to other entities.

9780170440943

Drawing context diagrams

There are two main notation styles used to represent context diagrams: Gane-Sarson and Yourdon-DeMarco. In this text, the Yourdon-DeMarco style is used.

When drawing context diagrams, the single process referring to the proposed system should be displayed in the centre of the diagram, with entities appearing on each side so that they are as balanced as possible. Each data flow should be represented as lines, normally curved, and must clearly indicate the direction the data is moving in. Labels on data flows that indicate which data is transferred should appear close enough to the line that it is clear which data flow the label is attached to.

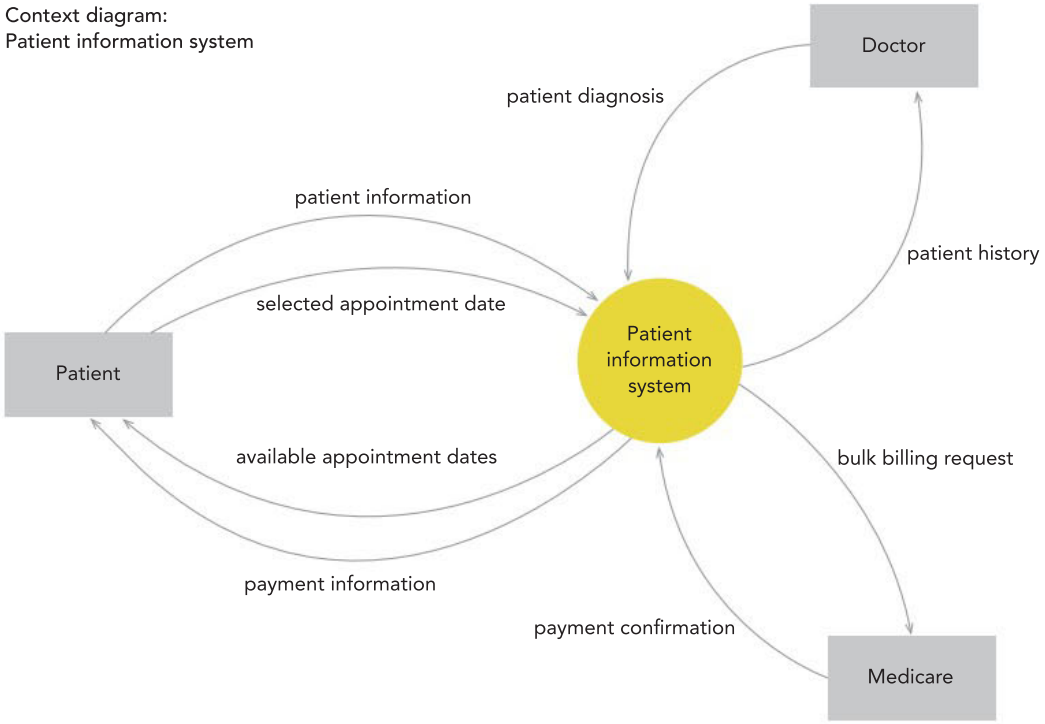


FIGURE 3.29 A context diagram for a patient information system

Data flow diagrams

A **data flow diagram (DFD)** is a graphical visualisation of the flow of information within a system, including data provided by external entities. DFDs provide more information than context diagrams, and are designed to show the data flowing in to and out from every function within the system. For this reason, DFDs are normally drawn with consideration to 'levels', where each level of the diagram contains more detailed information than the previous level. Context diagrams are therefore often referred to as 'Level 0 data flow diagrams', as they contain the least amount of information about the data flowing in to and out from the system.

Level 1 DFD diagrams contain the core processes within the system. There may be more than one of this level of diagram, depending on system complexity. The general rule

9780170440943

is that a DFD should contain at least three but no more than seven to nine processes; if there are more, they should be separated into related collections of processes across multiple DFDs. Level 2 DFDs (and higher) contain a more detailed look at the inner workings of the processes that were included in Level 1 DFDs. These are used when processes are quite complex, and more detail is needed to show what the processes are doing. If two or more levels of DFDs are created, the data that flows into and out of Level 1 DFDs must be the same; the only change in the diagrams is the level of detail shown for selected processes. The decision to proceed beyond Level 1 for DFDs can be separately made for each process, with the developer deciding when the level of detail is sufficient.

Much like context diagrams, DFDs contain processes, entities and data flows and they also show data stores, which represent where data is coming from.

Process

A **process** in a DFD is not the same as a process in a context diagram. While a context diagram process represents the whole system, a process within a DFD represents a whole function. For this reason, DFDs typically include at least as many processes as there are use cases in the system's use case diagram.

Processes in DFDs are shown using a circle, with the name of the process contained within that circle. Process names are typically short and begin with a strong verb and a singular noun, such as 'validate PIN' or 'print receipt', and normally represent the functions of the proposed system. Much like use cases, the name of a process should not be longer than two or three words.

In a DFD, processes can be connected to external entities, other processes and data stores. These are all connected via labelled data flows.

A process must have at least one input data flow and one output data flow. Processes must also transform data – the data going in should not be exactly the same as the data coming out.

Entity

A DFD may contain one or more entities, which, as in context diagrams, are the users or external systems that interact with the system being created. These are drawn as rectangles within the DFD, with the name of the entity contained within the rectangle. Much like actors in use case diagrams, entities should not be labelled with the names of real people, but instead with the abstract role those people have.

In a DFD, entities can only be connected to processes via a data flow. They cannot directly interact with data stores or other entities.

Data flow

A data flow represents a single piece or logical collection of data as it moves between entities, processes and data stores within the system. Much like in a context diagram, data flows in a DFD are represented by solid lines, typically curved, with an open arrow head representing the direction of the data flow. While most data flows are unidirectional, there are instances where data flows in both directions. In these cases, both ends of the data flow should show open arrow heads.

In a DFD, data flows can connect entities to processes, processes to other processes, and processes to data stores. They must always be coming from or going to a process and cannot directly connect data stores with each other.

9780170440943

Data store

A **data store** represents a collection of data that is stored in some way. Some examples of data storage include a database, a plain text file and an XML file. Data that flows out of a data store indicates that it is retrieved from it, whereas data flowing into the data store is assumed to be updating or adding to it.

In a DFD, data stores are represented by two parallel lines, with the name of the data store in between them. Data stores can only be connected to processes. Each data store should have at least one input and one output data flow.

Data store

FIGURE 3.30
A data store

Drawing data flow diagrams

Much like context diagrams, there are two main notation styles used to represent DFDs: Gane-Sarson and Yourdon-DeMarco. In this text, the Yourdon-DeMarco style is used.

DFDs should be drawn to reduce the amount of data flow overlaps. DFDs do not need to show entities if they have already been shown in a preceding level DFD, although they can still be included for clarity.

When creating a DFD, the following questions should be asked:

- Where does the data come from? Does it come in as input from an entity, or does it come from a data store?
- What happens to the data once it enters the system? Which process(es) does it flow into? Which data stores does it flow into?
- Where does data go once it enters a process? Is it stored in a data store? Is it returned to an entity as output?

The first step in constructing a DFD is to identify the processes that are needed to perform the work within the system. These are often closely related to the use cases that were included in the use case diagrams constructed in the analysis stage. At a minimum, each use case should be represented in a DFD as a process. Some use cases will require more than one process.

Once the processes have been identified they should first be connected to the relevant entities that provide the data that flows into the process. They should then be connected to other related processes and data stores, with each data flow appropriately labelled. Finally, any outputs from the processes should be included. In general, a process requires at least one data flow as input and at least one as output; if it has less than this, it is not a valid process.

Once all processes and DFDs have been created at Level 1, a decision can be made about whether any process within those diagrams requires further detail. If this is the case, a Level 2 DFD should be created to reflect that detail.

The layout of a DFD follows similar conventions to a context diagram, but as there are more processes involved, it is sometimes difficult to keep them all centred. The general rule is that processes are grouped in the middle of the DFD. Data flows should, as much as possible, be separated so they do not overlap other data flows, and data stores should be beneath the processes that use them. If entities are shown, they should be on the far left or far right of the diagram.

Once a DFD is complete, it should be checked for accuracy. Verifying that the DFD is accurate involves checking each process, data flow, data store and entity.

9780170440943

Each *process* should be checked to make sure it has a unique name, and that its name is a strong verb phrase and is sufficiently descriptive. Each process must also have at least one input data flow and one output data flow. Output data flows should have different pieces or collections of data from the input data flows. The DFD should not be overcrowded with processes.

Each *data flow* should be checked to make sure it is named after a piece of data or collection of data; they should be noun phrases. Each data flow should connect to at least one process. They should have a minimum amount of overlap with other data flows; it is preferred that there is no overlap.

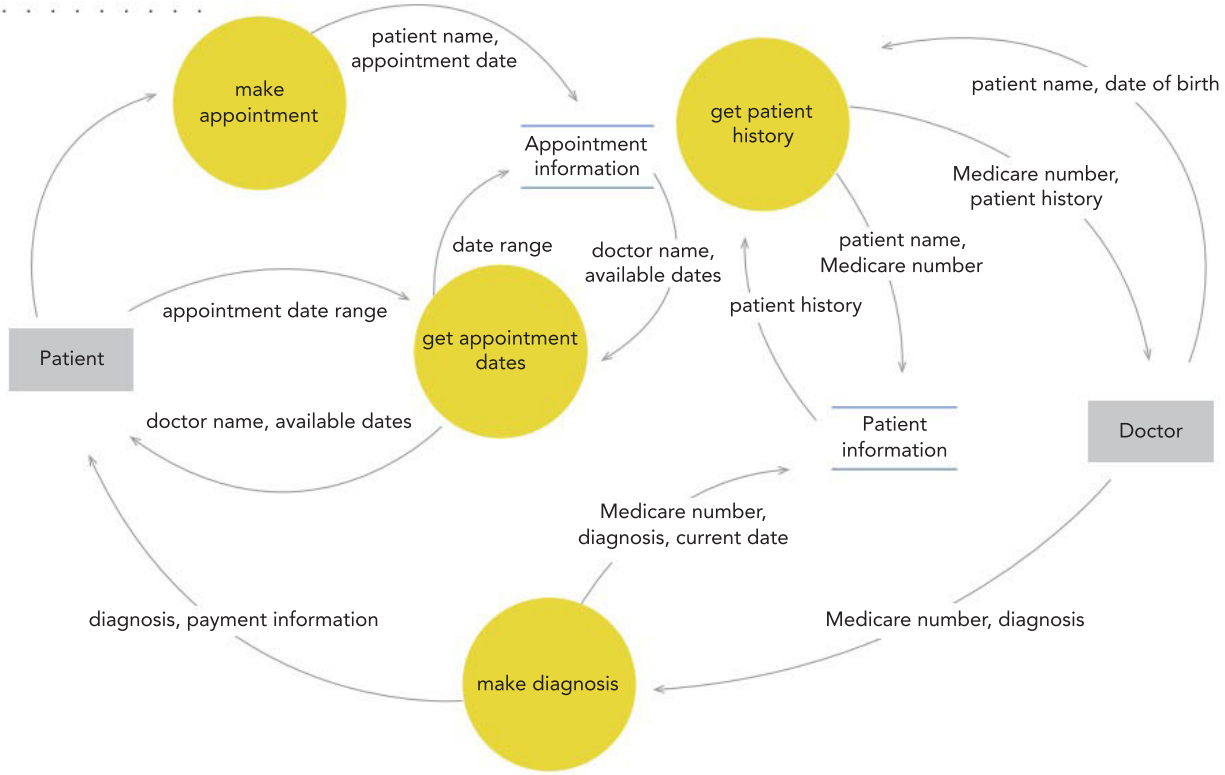


FIGURE 3.31 DFD Level 1 for a patient information system

Each *data store* should be checked to make sure it has a unique name and that it is named appropriately to the data it represents; the names should be noun phrases. Each data store should have at least one data flow connecting it to a process.

Each external *entity*, if shown, should match the name of an entity in the context diagram. The name of the entity should be unique and be a noun or noun phrase. Each entity should be connected to at least one process by at least one data flow.

9780170440943

CASE STUDY

Software to track music performances

Susan is a music director who is in charge of running music performances. She would like to replace her current manual system of tracking music performances with a software system so she can track concerts and performances without needing to rely on written notes. She would like the system to run on her computer at home as well as on her mobile phone. Susan is a relatively inexperienced technology user, but needs to be able to learn to use the software quickly; her preference is that it will require no more than an hour of training.

The system Susan needs must enable her to add performances and concerts. If she is adding a concert, she would also like to be able to assign a theme to it. She wants concert themes to come from a saved list that she can add to as required. Susan also needs the software to track the themes of each concert that she runs. She runs these twice a year, and must not use the same theme in any given six-year period; most people will not pay to see a concert again if the theme is the same. She tends to run her concerts in different locations on a six-year rotation.

Susan also needs the ability to edit concerts, performances and themes. In the future she would like the ability to upload recordings of live performances and concerts to the system.

She would like the software to be finished before her next concert in January next year.

Susan has approached a local software developer, Donna, and asked her to create the system for her. Donna began writing a software requirements specification (SRS) that outlined the functional requirements, non-functional requirements, constraints and scope of the system, which are listed below.

The functional requirements are:

- Music performances and concerts can be added (Figure 3.32).
- Music concert themes can only be attached to a concert if no concert has had that theme in the last six years (Figure 3.32).
- Music performances and concerts can be edited (Figure 3.33).
- Concert themes can be added (Figure 3.34).
- Concert themes can be edited (Figure 3.35).

3.1.1. Add Performance

This screen will allow the entering of performance information. Information consists of the performance name, date and theme, as well as a unique performance ID (numeric). Also recorded is whether the performance is a concert, and, if so, the theme of that concert.

Inputs:

- Performance ID (unique)
- Performance Name, Performance Date
- Whether it is a concert or not
- Concert Theme

Sequence of Operations:

- Input will come in from the User Interface, all items except concert theme are required fields
- Themes should be retrieved from a file containing all available themes so a user can select one
- No two concerts should have the same ID
- A theme only needs to be selected if it is a concert
- No concert should be added if a concert with the same theme has occurred in the last six years
- Once data is entered and validated, it should be saved to a performance file

Outputs:

A pop-up box should be displayed showing the success or failure of saving the concert information. Once information is saved, the user should be returned to the main menu.

FIGURE 3.32 An 'add performance' functional requirement for Susan's music performance system

9780170440943

The non-functional requirements are:

- Portability: the software must run on Susan's mobile phone as well as her computer at home.
- Usability: Susan should take less than an hour to learn how to use the software.

The constraints are:

- Technical: Data entry must be supported via keyboard as well as touch screen.
- Usability: Susan has very little experience using technology.
- Economic: Susan wants the software finished before January next year.

3.1.2. Edit Performance

Given a list of all performances, allow a user to change the information of any particular performance.

Inputs:

- Performance ID

Sequence of Operations:

- The user should be able to select a performance from a list of performances that have been saved to file.
- From this, they should be taken to a screen that looks like **3.1.1. Add Performance**, with the information pre-filled into the screen.
- They should be able to change the performance name, date and theme and whether it is a concert or not, but not the performance ID.
- These changes should be saved to the performance file once the save button is pressed.

Outputs:

A pop-up box should be displayed showing success or failure of saving the updated performance information. Once information is saved, the user should be returned to the main menu.

FIGURE 3.33 An 'edit performance' functional requirement from Susan's music performance system

3.1.3. Add Theme

This screen will allow the entering of theme information. Information consists of the theme name as well as a unique theme ID (numeric).

Inputs:

- Theme ID (unique)
- Theme Name

Sequence of Operations:

- Input will come in from the User Interface, all items are required fields.
- No two themes should have the same ID, so this should be validated before saving.
- Once data is entered and validated, it should be saved to a theme file.

Outputs:

A pop-up box should be displayed showing the success or failure of saving the theme information. Once information is saved, the user should be returned to the main menu.

FIGURE 3.34 An 'add theme' functional requirement from Susan's music performance system

9780170440943

System components that are in scope:
- the ability to add and edit music performances and music concerts
- the ability to add and edit concert themes.

System components that are out of scope:
- the ability to delete music performances, concerts or themes
- the ability to upload live recordings of performances and concerts.

Using the information already contained within the SRS, Donna must complete the SRS by creating a use case diagram, a context diagram and relevant data flow diagrams to represent the system.

3.1.4. Edit Theme

Given a list of all themes, allow a user to change the information of any particular theme.

Inputs:
- **Theme name**

Sequence of Operations:
- **The user should be able to select a theme from a list of themes that have been saved to file.**
- **From this, they should be taken to a screen that looks like 3.1.3. Add Theme, with the information pre-filled into the screen.**
- **They should be able to change the theme name but not the theme ID.**
- **These changes should be saved to the theme file once the save button is pressed.**

Outputs:

A pop-up box should be displayed showing the success or failure of saving the changed theme information. Once information is saved, the user should be returned to the main menu.

FIGURE 3.35 An 'edit theme' functional requirement from Susan's music performance system

Step 1: Creating a use case diagram

Determine the actors

As Susan is the only person who will use this system, there is only one actor. Susan's role in relation to the system is *Music Director*.

Determine the use cases

To create a use case diagram, Donna looks at the functional requirements of Susan's system:
- Music performances and concerts can be added (Figure 3.32).
- Music concert themes can only be attached to a concert if no concert has had that theme in the last six years (Figure 3.32).
- Music performances and concerts can be edited (Figure 3.33).
- Concert themes can be added (Figure 3.34).
- Concert themes can be edited (Figure 3.35).

Most of these functional requirements can be represented as a use case. These need to have short names that begin with strong verbs:
- Add performance
- Add concert
- Edit performance
- Edit concert
- Add theme
- Select theme
- Edit theme

9780170440943

After writing out the list of potential use cases, Donna realises that concerts are just a particular type of performance, the only difference being that a concert requires that a theme be selected, while a performance does not. She revises her list of use cases as a result:

- Add performance
- Edit performance
- Select theme
- Add theme
- Edit theme

Determine the relationships

In this scenario, concerts are special types of performances, and not all performances are concerts. This suggests that the relationship between a performance and a concert is an *extension*, where a concert is an extension of a performance. The difference between the two is that a concert requires that a theme is selected, while a performance does not. Therefore, the *select theme* use case has an *extend* relationship with the *add performance* use case.

Determine the system boundary

All of the listed use cases are within scope of the functional requirements of the new software. This means the system boundary will include all use cases. Donna could also choose to include the functional requirements that are out of scope in the use case diagram, such as the ability to upload live recordings, but this is optional.

Draw the use case diagram

Given all of the information she now has about the actors, use cases, relationships and system boundary, Donna is able to draw the use case diagram, as shown in Figure 3.36.

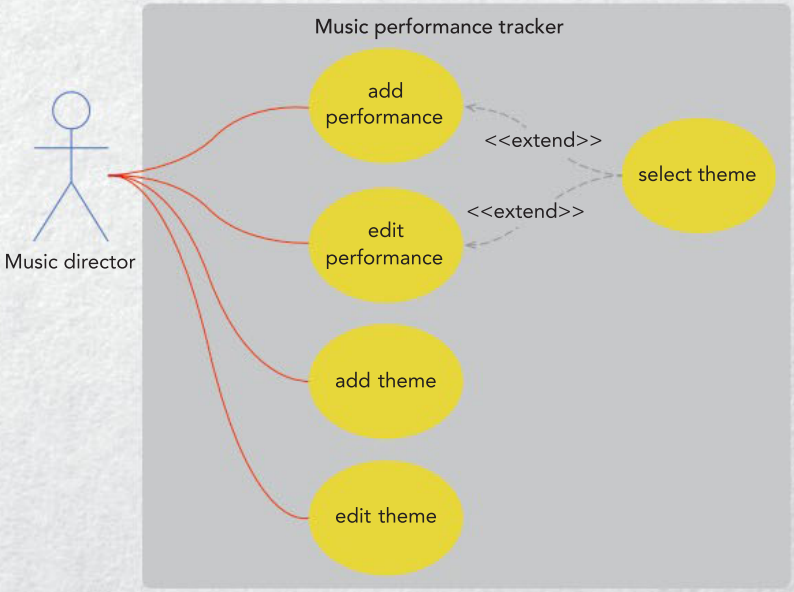


FIGURE 3.36 A use case diagram for the Music Performance Tracker software

It is also possible to combine the 'add performance' and 'edit performance' use cases, as well as the 'add theme' and 'edit theme', as the functionality inside them is very similar. For the purposes of this example, however, they have been left in.

Step 2: Creating a context diagram

Determine the processes

The single process shown in the context diagram represents the whole system: *Music Performance Tracker*.

Determine the entities

Donna checks the use case diagram to see which actors are involved in the system; the only actor is the *Music Director*. As no data is received from any other external sources, this must be the only entity that interacts with the music performance software.

9780170440943

Determine the data flows

To determine the data that flows in and out of the music performance system, Donna checks the use case diagram again for each of the use cases:

- Add performance
- Edit performance
- Select theme
- Add theme
- Edit theme

Each of these must provide data in some way to the system, and some of them must return data back to the music director. Donna systematically considers each use case to decide if it must be included in the context diagram. All of them except the *select theme* use case require data from the user, so she uses the information in the functional requirements to determine the following data flows:

From the *Music Director* to the *Music Performance Tracker*:

- Add performance: performance details (performance ID, performance name, date, whether it is a concert, theme name)
- Edit performance: performance ID, changed performance details
- Add theme: theme details (themeID, theme name)
- Edit theme: theme name, changed theme name

From the *Music Performance Tracker* to the *Music Director*:

- Add performance: Theme re-used error
- Add performance: Performance added success or fail message
- Edit performance: Theme re-used error
- Edit performance: Performance edited success or fail message
- Add theme: Theme added success or fail message
- Edit theme: Theme edited success or fail message

Draw the context diagram

Given all of the information she now has about the process, entities and data flows, Donna is able to draw the context diagram, as shown in Figure 3.37.

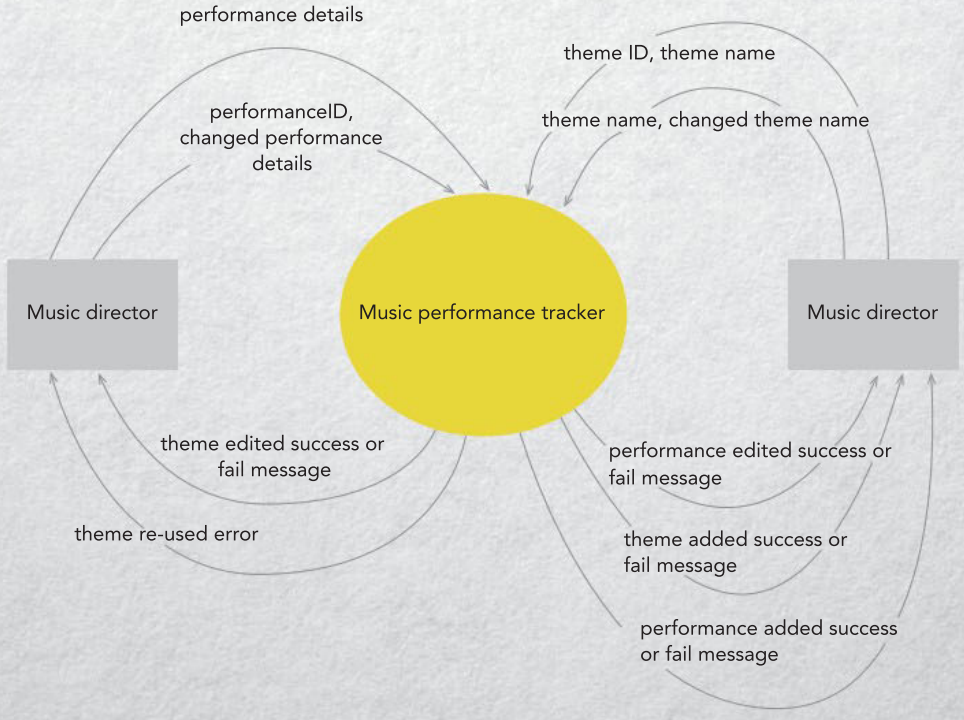


FIGURE 3.37
A context diagram for the Music Performance Tracker software

SCHOOL-ASSESSED TASK TRACKER

☑ Project plan	☑ Justification	☑ Analysis	☐ Folio of alternative designs ideas	☐ Usability tests	☐ Evaluation and assessment	☐ Final submission

9780170440943

Step 3: Creating data flow diagrams

Determine the processes

Donna uses the use case diagram again to determine the processes that must exist in her data flow diagram. Each of the use cases must have a corresponding process in the data flow diagram:

- Add performance
- Edit performance
- Select theme
- Add theme
- Edit theme

Determine the entities

Donna checks the context diagram and only a single entity is needed: *Music Director*.

Determine the data stores

Before determining which data flows are needed, Donna must consider where data will be stored. Performances, including concerts, can be stored in a single data store. Themes should be stored in a different data store, as they are a different collection of data to the performance data.

Donna therefore decides that there will be two data stores: *Performances* and *Themes*.

Determine the data flows

Each of the processes must now be considered in relation to the data they receive from external entities, the data they pass to other processes and the data they store and retrieve from the data stores.

Donna checks the context diagram to look at each of the data flows she represented going to the system. This takes quite a while, as she must make sure she shows the movement of every piece of data to and from processes, data stores and entities.

TABLE 3.1 Data flows going into and out of the music performance system

	From	To	Data	Reason
1	Add performance	Music Director	Performance added success or fail message	Output to the Music Director when adding a performance
2	Add performance	Music Director	Theme re-used error	Output to the Music Director if re-using a theme before 6 years is up
3	Add performance	Performances data store	Performance details	Sending the performance data to be saved
4	Add performance	Select theme	Theme name	Sending the theme name to get a valid theme ID
5	Add theme	Music Director	Theme added success or fail message	Output to the Music Director when adding a theme
6	Add theme	Themes data store	Theme ID, theme name	Sending a new theme and theme ID to the data store to be saved
7	Edit performance	Music Director	Performance edited success or fail message	Output to the Music Director when editing a performance

9780170440943

	From	To	Data	Reason
8	Edit performance	Music Director	Theme re-used error	Output to the Music Director if the changed theme was used in the last 6 years
9	Edit performance	Performances data store	Changed performance details	Sending the changed performance details to be saved
10	Edit performance	Performances data store	Performance ID	To retrieve the performance details when a performance is selected
11	Edit performance	Select theme	Theme name	Sending the theme name to get a valid theme ID
12	Edit theme	Music Director	Theme edited success or fail message	Output to the Music Director when editing a theme
13	Edit theme	Themes data store	Theme ID, changed theme name	Sending the changed theme details to be saved
14	Edit theme	Themes data store	Theme name	To retrieve the theme ID when a theme is selected to be edited
15	Music Director	Add performance	Performance details	Providing performance details to be saved
16	Music Director	Add theme	Theme details	Providing theme details to be saved
17	Music Director	Edit performance	Performance ID, changed performance details	Providing changed performance details to be saved
18	Music Director	Edit theme	Theme name, changed theme name	Providing changed theme details to be saved
19	Performances data store	Edit performance	Performance ID, performance details	Checking to see if a performance exists and returning its details if it does
20	Select theme	Themes data store	Theme name	Sending the theme name to get a valid theme ID
21	Themes data store	Edit theme	Theme ID, theme name	Providing theme details to be edited
22	Themes data store	Select theme	Theme ID	Providing a theme ID for a given theme name

9780170440943

Draw the data flow diagram(s)

Donna decides that she will only use one data flow diagram, as there are only five processes. Given all of the information she now has about the process, entities, data stores and data flows, she draws this data flow diagram, as shown in Figure 3.38.

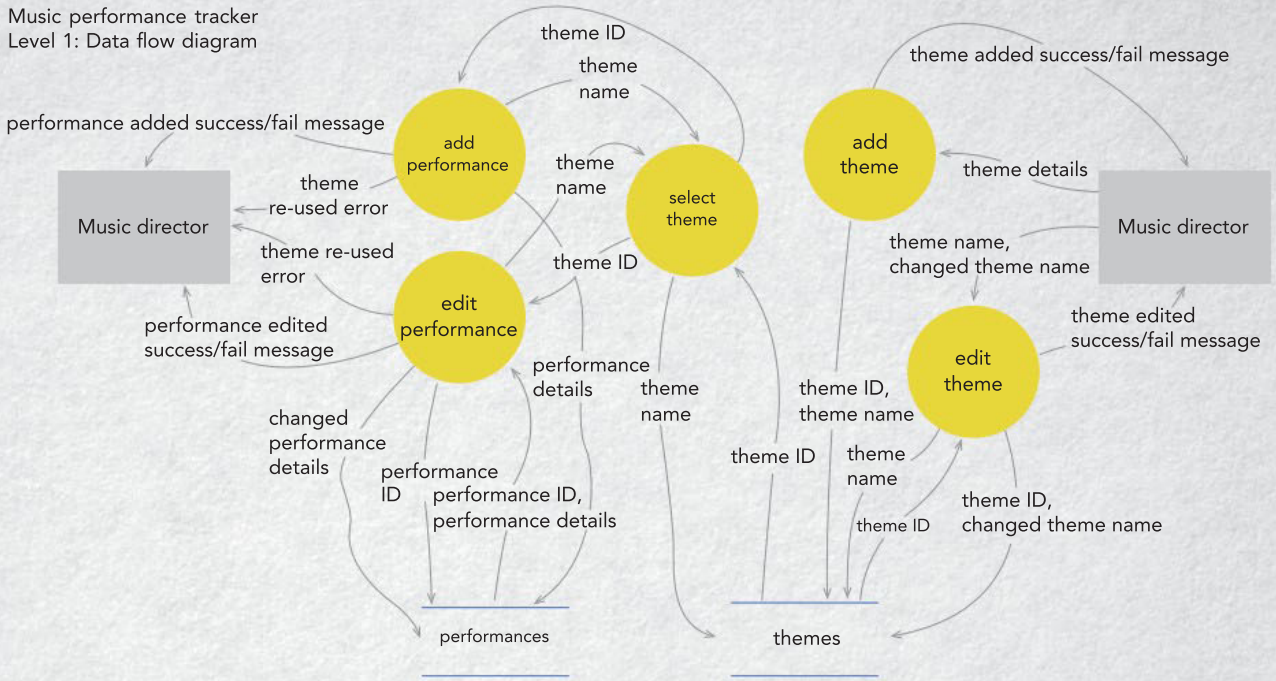


FIGURE 3.38 A data flow diagram for the Music Performance Tracker software

Security considerations

In an increasingly connected world, security considerations for software have become paramount for any business, large or small. The risk of a software solution being compromised is very high, with data breaches having widespread repercussions ranging from loss of reputation to financial loss and possible violation of legal regulations. For example, some countries hold businesses legally responsible for a system that does not comply with laws regarding the storage and communication of electronic data. Programmers therefore have an obligation to protect the security of data within a system as much as possible. This can involve implementing security features such as encryption for data storage and transfer, as well as putting in place authentication protocols to access elements of the software or the software solution as a whole.

Encryption

Encryption is the process by which **plain text data** is encoded – scrambled – so that it is unreadable by unauthorised applications or people. Once encrypted, this data is referred to as **cipher text data**. Plain text data is encrypted using a key, and the resulting cipher text data can only be decrypted by a person or application that has a decryption key. Encryption is typically used to protect data when it is stored on a computer system, as well as to protect data as it is transferred over unsecured networks.

A common application of encryption is to encrypt messages using a digital signature. Digital signatures give people the confidence that the message they receive is authentic.

9780170440943

Shutterstock.com/Rawpixel.com

FIGURE 3.39 Encryption is a common method of securing data.

There are two main algorithms used to **encrypt** data. The first involves using **symmetric key encryption**, where the key used to encrypt the data is the key that is also used to **decrypt** that data. An analogy to describe this is when a code is needed to open a safe, and anyone who has that code can open the safe and access its contents.

Examples of popular symmetric key encryption algorithms are AES, Twofish, Blowfish, 3DES and RC4.

A second type of encryption algorithm is **public key encryption**, also known as **asymmetric key encryption**, where the key used to encrypt data is not the same as the key that is used to decrypt that data. In this algorithm, a **public key** is used to encrypt the data. This key can be used by any person or application and is generally widely known. The data can only be decrypted using a **private key**, which is known only to the recipient person or application. Public key encryption is used widely in the computing industry, such as with **Transport Layer Security (TLS)** and **Secure Sockets Layer (SSL)**. This type of encryption is typically used when data is transferred over an open networked environment, such as the Internet. A popular public key encryption algorithm is RSA, which is the standard used for data transfers via the Internet. Other public key encryption algorithms are Diffie-Hellman, ECC and DSA.

Implementing encryption algorithms

Encryption algorithms are quite complex to implement and it is critically important that the implementation is accurate and bug-free. A faulty algorithm could, at best, result in data not being able to be decrypted and, at worst, allow data to be decrypted easily by unauthorised people or applications.

Most programming languages have built-in functions or third-party packages that provide the ability to implement encryption without needing to write the encryption algorithm yourself. For example, Python has libraries that include implementations of encryption algorithms such as AES and RSA. Similarly, Visual Basic contains classes that implement 3DES and AES.

An example of how to represent encryption in pseudocode is shown in Figures 3.40 and 3.41.

THINK ABOUT SOFTWARE DEVELOPMENT 3.3

Bluetooth is commonly used to transfer data between systems, particularly mobile phones. Conduct research on which type of encryption algorithm is typically used for Bluetooth data encryption.

The AES encryption algorithm is one of the most popular symmetric encryption algorithms used in the computing industry. Many financial and government institutions use this type of encryption. AES encryption typically uses 128-bit keys, but stronger versions can also use 192-bit and 256-bit.

TLS and SSL are security protocols that are used over computer networks. These protocols are typically used by web browsers and email programs to provide encryption over the Internet. SSL is the predecessor to TLS; TLS was based on SSL 3.0. This means that TLS should be used over SSL for relevant applications unless the use of SSL is a constraint.

SCHOOL-ASSESSED TASK TRACKER

☑ Project plan	☑ Justification	☑ Analysis	☐ Folio of alternative designs ideas	☐ Usability tests	☐ Evaluation and assessment	☐ Final submission

9780170440943

```
ALGORITHM encryptData(plainTextData)
BEGIN
    ENCRYPT (plainTextData) using 3DES
END

ALGORITHM decryptData(encryptedCipherText)
BEGIN
    plainTextData ← DECRYPT(encryptedCipherText) using 3DES
    RETURN plainTextData
END
```

FIGURE 3.40 Representing symmetric key encryption in pseudocode

```
ALGORITHM encryptData(plainTextData, publicKey)
BEGIN
    ENCRYPT(plainTextData, publicKey) using RSA
END

ALGORITHM decryptData(encryptedCipherText, privateKey)
BEGIN
    plainTextData ← DECRYPT(encryptedCipherText, privateKey) using RSA
    RETURN plainTextData
END
```

FIGURE 3.41 Representing public key encryption in pseudocode

Authentication

Aside from encryption, another method of securing data to reduce the risk of data breaches is to integrate authentication functionality into a software solution. Authentication in computing can have two meanings: one is to prove an identity, the other is to prove that a user has a right to access a software system. For the purposes of VCE Software Development, authentication should be interpreted as the latter: authorisation.

Authentication methods can range from the simplest, single-factor authentication to more complex multi-factor authentication.

Single-factor authentication

This is the simplest method of authentication, where *something you know*, typically a **username and password**, is required to log in to a software solution. Once logged in, users can have **full access** or **restricted access** to the functionality within the software, depending on the level of access they have been granted. This method of authentication is increasingly considered inadequate due to the prevalence of password cracking tools available on the Internet.

Two-factor authentication

This type of authentication involves *something you know* as well as *something you have*. **Two-factor authentication**, also known as two-step verification, typically involves the user possessing some physical item alongside a password that they must use to access a particular piece of software. One example of two-factor authentication is the process you take to withdraw money from an ATM. To do this, you traditionally must have a physical card in your possession as well as a password, which in this instance is your PIN. Another common example of two-factor authentication is the use of a secondary application to provide a single-use verification code to the user that changes frequently, such as every 60 seconds. The user must provide this alongside a username and password to gain access to the relevant software.

Alamy Stock Photo/Cristian Dina

FIGURE 3.42 Verification codes are required for two-factor authentication.

Google Authenticator is an application that implements two-factor authentication for users. It uses a Time-based One-Time Password algorithm (TOTP) for authenticating users of mobile applications.

THINK ABOUT SOFTWARE DEVELOPMENT 3.4

Research Time-based One-Time Password (TOTP) algorithms. How do they work? Aside from Google, what other companies use this algorithm to authenticate users? How secure is this method of authentication?

The ease of implementing two-factor authentication is highly dependent on the programming language selected. Some languages have third-party packages and libraries that allow for 'plug-in' style inclusion into a software solution, such as Swift and Python, but this is not necessarily the case in all programming languages.

Multi-factor authentication

Multi-factor authentication typically involves a user providing three or more pieces of evidence to prove that they are who they say they are. Typically, these involve *something you know*, *something you have* and *something you are*. It can also involve *somewhere you are*. As in single-factor and two-factor authentication, *something you know* is typically a username and password and *something you have* is typically a physical or digital authenticator that acts as a secondary device providing single-use passwords. The third element, *something you are*, typically uses one or more physical characteristics of the user to authenticate them on the system, such as checking against **biometric data**. Biometric data is data that is obtained from humans, which can include fingerprints, iris scanning, facial recognition, palm prints, hand geometry and DNA matching. It can also involve behavioural characteristics, such as typing speed, key-press patterns, gait patterns and voice recognition. The fourth element, *somewhere you are*, involves location-based factors that involves the physical location of the user. For example, a user may only be able to access a system if they are on a hard-wired network within an organisation, or within a range of GPS coordinates.

Shutterstock.com/metamorworks

FIGURE 3.43 Biometric data can involve facial or voice recognition.

THINK ABOUT SOFTWARE DEVELOPMENT 3.5

What are some of the risks of two-factor and multi-factor authentication processes?

The complexity of implementing multi-factor authentication is tied to the complexity of the biometric data that is required for authentication to be successful. These methods are typically very expensive, and tend to be implemented on systems that are highly sensitive or require tight security.

SCHOOL-ASSESSED TASK TRACKER

☑ Project plan	☑ Justification	☑ Analysis	☐ Folio of alternative designs ideas	☐ Usability tests	☐ Evaluation and assessment	☐ Final submission

9780170440943

Next steps

In this chapter, we discussed the features of project management, and the need for regular monitoring and adjustment. A log or weblog (blog) of changes is to be kept throughout Unit 3 and Unit 4.

Next, we discussed the data collection methods available to analyse client requirements, scope of the solution, functional and non-functional requirements and the various constraints that can impact on a project.

Finally, we discussed data security, with regard to how it relates to you and your file management requirements.

Your next step, upon completion of the chapter summary, is to work towards completion and submission of the solution analysis for Unit 3, Outcome 2, according to your teacher's instructions.

As you collect your data, take steps to protect your respondents and subjects. You should begin to think about relevant constraints, the scope of the specification and appropriate analytical diagrams to represent data flow and users. Chapter 4 begins with a discussion of the software solution design requirements for Unit 3, Outcome 2.

9780170440943

3 CHAPTER SUMMARY

Essential terms

actor an entity that can interact with the software solution as shown in a use case diagram

analysis stage the stage of the problem-solving methodology where solution requirements, constraints and scope are determined

association a relationship between two elements in a use case diagram

asymmetric key encryption *see* **public key encryption**

biometric data data that is obtained from humans, which can include fingerprints, iris scanning, facial recognition, palm prints, hand geometry and DNA matching, as well as behavioural characteristics such as typing speed, key-press patterns, gait patterns and voice recognition

cipher text data data that has been encoded so that it is unreadable by unauthorised applications or people

clarity the extent to which a product is coherent and intelligible

close-ended questions questions that can be answered with a finite set of responses

concepts (project management) the milestones and dependencies within a project timeline

concurrently when a task is carried out at the same time as another task

constraints factors that may limit or restrict solution requirements

context diagram a visualisation of a system in its entirety that indicates the data that is passed into and out of the system

critical path the shortest possible time in which a project can be completed

data raw, unprocessed facts and figures

data flow the movement of a piece or collection of data within an information system, as shown in context diagrams and data flow diagrams (DFDs)

data flow diagram (DFD) a graphical visualisation of the flow of information within a system, including data provided by external entities

data store a representation of a collection of data that is stored in some way within a system

decrypt to decode encrypted cipher text data

design stage the stage of the problem-solving methodology where the function and appearance of a solution are planned, and evaluation criteria created

encrypt to encode plain text data so that it cannot be read by unauthorised applications or people

encryption the process of encrypting data

entity the users or external systems that interact with the system being created

evaluation criteria the benchmark or set of standards by which a solution or design is measured

event a special type of method that is called when an object's state changes

extend a relationship between use cases where one use case has optional or additional functionality, which is represented in a use case diagram as a second use case

fit for purpose to be well suited for a role or purpose

full access access to all of the functionality of a software solution without any restrictions

functional requirements the desired operations of a program that have specified inputs, behaviours and outputs

9780170440943

3 CHAPTER SUMMARY

functionality the extent to which a solution is suited to its purpose

Gantt chart shows the progress of a project by placing tasks on a timeline, often with comments or annotations

generalisation a parent–child relationship between two elements in a use case diagram

include a relationship between use cases where one use case is tied to, or relies upon, the functionality contained within another use case

interview a face-to-face meeting between people for consultative purposes

maintainability how easy a solution is to look after once it has been put in place

non-functional requirements qualitative requirements of a solution, often tied to solution constraints

observation a method of data collection that involves physically observing how a system operates and how it is used

open-ended questions questions where the number of potential answers is infinite

plain text data data that can be read without any manipulation

portability how easily a solution is able to be used in different operating environments

predecessor a task that must be completed before another one can be performed

private key an encryption key used in public key encryption that is only known to the recipient person or application

problem-solving methodology (PSM) an approach that develops the stages involved in solving a problem

process (context diagram) an abstract representation of the whole system being created

process (data flow diagram) an abstract representation of a function within a system

processes (project management) task identification, sequencing and allocation of time and resources within a project timeline

project management a method of recording the progress of a project and managing resources to operate within time, resource and cost availability

public key an encryption key used in public key encryption that is known by any person or application

public key encryption a type of encryption where the key used to encrypt the data is different to the key that is used to decrypt that data

qualitative data data that consists of descriptive details, usually gathered via surveys or interviews

quantitative data data that can be easily processed in a statistical manner, usually composed of definite numbers

relationship the connections between elements within a use case diagram

reliability how much a solution can be depended upon to function as designed, and for how long

report a written document providing a summary or finding in relation to the context or system being analysed

restricted access access to functionality within a software solution is limited or restricted based on user or group permissions

robustness how well a software solution responds to errors that occur when the software is being used

Secure Sockets Layer (SSL) an obsolete security protocol designed to provide secure transfer of data over computer networks

9780170440943

slack time the length of time that a task can run overtime before affecting other tasks

software programs used by a computer

software developer a human who participates in design and creation of software programs, typically by writing programming code

software requirements specification (SRS) a single document that contains the outcomes of the analysis stage of the problem-solving methodology, including scope, constraints, functional requirements and non-functional requirements

successor a task that must be completed after another task

survey a set of questions that ask for a response to be selected from a list of alternatives

symmetric key encryption a type of encryption where the key used to encrypt the data is also the key that is used to decrypt that data

system boundary a rectangle around relevant use cases that indicate the use cases that are within the scope of the solution

Transport Layer Security (TLS) a security protocol designed to provide secure transfer of data over computer networks

two-factor authentication verification that involves users possessing two forms of information to confirm their identity

Unified Modellign Language (UML) a general-purpose visual modelling language

uptime the time during which a machine, solution or application is operational

usability the extent to which a system is easy to learn and use

use case a representation of the transactions or functions a user (actor) can complete in a system, as shown in a use case diagram

use case diagram (UCD) a method of describing how a user interacts with a system, using Unified Modelling Language (UML)

username and password a username is a name that uniquely identifies a person in a software solution; a password is a secret word, phrase or set of characters that allows that person access

work breakdown structure (WBS) an often hierarchical breakdown of a project that organises the work to be done into manageable sections, often displayed as a visual outline or map

Important facts

1. **Project management** is the practice of **applying techniques, processes, tools, knowledge** and **skills** to deliver a solution. Features of **project management** include **identification of tasks, sequencing, time allocation, milestones, dependencies** and **critical path**.
2. **Critical path** indicates the shortest time possible to complete the project.
3. If there is any change on the critical path, the timing of the entire project is affected.
4. **Data collection** involves **surveys, interviews, reports** and **observations**.
5. A **survey** is a set of questions that ask for a response from a user.
6. **Close-ended questions** in surveys are where the number of responses are finite.
7. **Open-ended questions** in surveys are where the number of responses are infinite.
8. **Quantitative data** contains information that is easily collated, such as values, numbers or counts; typically this involves numeric variables.
9. **Qualitative data** contains information that is not easily measured, such as opinions and qualities.
10. **Surveys** are relatively inexpensive, and results can be immediately collected.
11. Processing survey results can take time if open-ended questions are used.
12. People are not always truthful when answering surveys.
13. **Interviews** are usually conducted face to face, and can be one-on-one or in groups.

9780170440943

3 CHAPTER SUMMARY

14 Interviews can take a lot of time to complete, but allow in-depth data to be collected.

15 **Interviews** are very useful for eliciting **feelings, attitudes, judgements** and **opinions** that are too complex to easily record in a survey.

16 **Reports** are written documents providing summaries or findings in relation to a system being analysed.

17 Reports include **error reports, customer complaint summaries, uptime reports and system performance reports**.

18 The types of reports collected will vary depending on what is most relevant for the proposed system.

19 Reports are useful as they are often pre-prepared, which can save time and money.

20 A disadvantage of reports is that the data may have been manipulated to present a particular point of view.

21 Using data from reports can be risky if the source of those reports is not reliable.

22 **Observations** involve physically observing how a system operates and is used.

23 Observations are considered **unbiased**, as information can be gathered without asking for an opinion or judgement.

24 Observations can be **time-consuming**, and they may not always provide the information needed due to being performed at an inopportune time.

25 **Functional requirements** are part of the solution required and directly relate to what a solution will do.

26 Functional requirements are typically described in terms of **required inputs, sequence of operations** and **expected outputs**.

27 **Non-functional requirements** are qualities and aspects of the solution that are desired but do not affect what the solution does.

28 Non-functional requirements are often tied to the constraints of the system.

29 Non-functional requirements are described in terms of **usability, reliability, portability, robustness** and **maintainability**.

30 Determining non-functional requirements most often involves client interviews and observations.

31 **Usability** relates to how easy a system is to learn and use.

32 **Reliability** relates to how much, and for how long, a system can be depended upon to function as designed.

33 **Portability** relates to how easily software can be used in different operating environments.

34 **Robustness** relates to how well a software solution responds to errors that occur while the software is being used.

35 **Maintainability** relates to how easy the software is to look after once it is put in place.

36 A **software requirements specification (SRS)** is a single document that outlines all of the elements considered in the analysis stage: **constraints, scope, functional requirements** and **non-functional requirements**.

37 An SRS provides all of the required information about the proposed system needed in order to design the system.

38 An SRS should contain a cover page, table of contents, numbered sections, headings and subheadings, page numbers and relevant appendices.

39 Three methods of depicting interfaces between solutions are **use case diagrams, context diagrams** and **data flow diagrams**.

9780170440943

40 A **use case diagram** visually describes how a user interacts with a system.

41 Use cases include **actors** who have relationships with use cases.

42 An actor represents an entity that can interact with the functionality within software.

43 Actors should be described in terms of roles rather than real people.

44 Actors are represented as stick figures in use case diagrams.

45 Use cases describe transactions or functions an actor can complete on a system.

46 Use cases are drawn using an ellipse with the name of the use case written inside.

47 Use cases should be described using strong verbs to describe the action or function being represented.

48 **Relationships** represent the connections between elements in a use case diagram.

49 Relationships can exist between actors and use cases, actors and other actors, and between two use cases.

50 **Associations** are the standard form of representing a relationship.

51 Associations are represented as solid, straight lines.

52 Generalisations are a type of relationship that is considered parent-child.

53 **Generalisations** are represented as solid, straight lines with a closed arrow head pointing from child to parent.

54 **Include** and **extend** are special types of relationships in a use case diagram.

55 The *include* relationship represents a use case that is entirely included in another use case.

56 The *extend* relationship represents a use case that provides additional and/or optional functionality within a system that is connected to a use case, but not always run.

57 Include and extend relationships are represented with dashed or dotted lines and open arrow heads pointing to the relevant use case.

58 System boundaries make clear what is included within a system and what is not.

59 System boundaries are useful to show the scope of a system.

60 Context diagrams provide a visualisation of data that is passed into and out of a system.

61 Context diagrams are brief and do not show much detail, focusing only on the flow of data.

62 Three primary components of a context diagram are processes, entities and data flows.

63 A context diagram process is an abstract representation of the whole system.

64 A context diagram entity is a user or external system that interacts with the system being described.

65 A context diagram data flow represents a single piece or logical collection of data as it moves into and out of the system represented.

66 Two **notation styles** used to represent context diagrams and data flow diagrams are Gane-Sarson and Yourdon-DeMarco.

67 **Data flow diagrams (DFDs)** are visualisations of the flow of information within a system, including data provided by external entities.

68 DFDs provide more information than context diagrams, as they show all of the processes that occur within a system.

69 Four primary components of DFDs are **processes, entities, data flows** and **data stores**.

70 A **DFD process** represents a function or method within the system.

71 A **DFD** entity represents a user or external system that interacts with the system.

72 A **DFD data flow** represents a single piece or logical collection of data as it moves between entities, processes and data stores within the system.

73 A **DFD data store** represents a collection of data that is stored in some way within the system.

74 The repercussions of **data breaches** include loss of reputation, financial loss and possible violation of legal regulations.

75 Programmers are obliged to protect the security of data as much as possible within a software solution.

9780170440943

3 CHAPTER SUMMARY

76 **Encryption** is the process by which plain text data is encoded so that it cannot be read by unauthorised applications or people.

77 Once encrypted, data is referred to as **cipher text data**.

78 Encryption typically involves a **key that encrypts** the text, and a **key that decrypts** the resulting cipher text.

79 Encryption keys are the same as decryption keys in **symmetric key encryption**.

80 Encryption keys are different to decryption keys in **public key encryption**.

81 **Transport Layer Security (TLS)** and **Secure Sockets Layer (SSL)** are protocols designed to provide secure transfer of data over computer networks.

82 TLS is widely used over SSL; SSL is considered obsolete.

83 Most programming languages have built-in functions, third-party packages or plug-ins that provide the ability to encrypt and decrypt data.

84 Another method of securing data is to use authentication.

85 **Authentication** is also known as authorisation.

86 **Single-factor authentication** typically relies on user knowledge of a username and password.

87 **Two-factor authentication** uses two methods of authorising a user, such as knowing a password and possessing a swipe-card authenticator.

88 **Multi-factor authentication** involves three or more methods of authorising a user, such as using a password, possessing an authenticator and using biometric data or location mapping.

89 **Biometric data** includes fingerprints, iris scans, facial recognition, DNA matching, palm prints, hand geometry.

90 **Behavioural characteristics** include typing speed, gait patterns, key press patterns and voice recognition.

91 Implementing multi-factor authentication is as complex as the elements that are used to authenticate; implementation of fingerprint scanning would be simpler than DNA matching, for example.

9780170440943

TEST YOUR KNOWLEDGE

Review quiz

What is a software solution?

1 What is meant by a software solution?

Project management

2 What is meant by project management?
3 Identify two consequences of a badly managed project.
4 Why is a Gantt Chart used?
5 Differentiate between concepts and processes in project management.
6 Differentiate between predecessors and successors on a Gantt Chart.

Collecting data

7 How does a survey differ from an interview?
8 When would an observation be a preferred method of collecting data?
9 What are three advantages of an interview?
10 What are two disadvantages of using reports as part of data collection?

Functional and non-functional requirements

11 How is a functional requirement different to a non-functional requirement?
12 Categorise each of these requirements as functional or non-functional:
   a A report must print to a printer.
   b All font colours must be green.
   c A discount must be applied to a product.
   d The drone must be able to navigate a path through a maze.
   e The body mass index of a person will be calculated.
   f A typical six-year old should understand all of the words displayed.
   g Button sizes must be big enough for touch-screen capability.
   h Input can be via voice or keyboard.
13 What is the difference between reliability and robustness?
14 What does it mean for a software solution to be maintainable?
15 What does it mean for a software solution to be usable?
16 Explain portability in terms of non-functional requirements.

SCHOOL-ASSESSED TASK TRACKER

☑ Project plan	☑ Justification	☑ Analysis	☐ Folio of alternative designs ideas	☐ Usability tests	☐ Evaluation and assessment	☐ Final submission

9780170440943

TEST YOUR KNOWLEDGE

Software requirements specifications

17 Why are software requirements specifications written?

18 What is contained within an SRS?

Interfaces between solutions, users and networks

19 What is the purpose of a use case diagram?

20 What is the purpose of a context diagram?

21 What is the purpose of a data flow diagram?

22 Explain the difference between *include* and *extend* in a use case diagram.

23 Explain the difference between an *association* and a *generalisation* in a use case diagram.

24 How does a *process* differ between context diagrams and data flow diagrams?

25 A proposed food ordering system has two types of users, a chef and a server. The server places food orders given to them by customers. A chef confirms orders and flags them as cooked so a server knows they can take the food to the customer. Once a server has delivered the food to the customer, they remove the order from the queue.

- **a** Draw a use case diagram to represent this system.
- **b** Draw a context diagram to represent this system.
- **c** Draw a data flow diagram to represent this system.

Security considerations

26 What is the difference between symmetric key encryption and public key encryption?

27 Which is best to use for security on a web server, TLS or SSL? Why?

28 What type of authentication involves entering a username and password?

29 When would it be advisable to use two-factor authentication over single-factor authentication?

30 When would it be advisable to use multi-factor authentication over two-factor authentication?

9780170440943

APPLY YOUR KNOWLEDGE

1. Consider the problem, opportunity or need you have selected for your School Assessed Task (SAT).
2. Construct a Gantt chart with tasks, milestones and dependencies, including predecessors and successors.
3. Ensure you have collected all relevant data for your solution. Your constraints and scope should have been included in your design brief, but if not, make sure you have documented the following:
   a. Constraints: ensure you have considered technical, economic, social, legal and usability constraints.
   b. Scope: ensure you have considered the constraints of your system when deciding what is in scope and what is out of scope.
4. In your software requirements specification, include all elements related to the constraints and scope that were outlined in your design brief. Expand on any element where required.
5. Collect data in order to determine the functional and non-functional requirements of your chosen system. This will likely involve client and/or user interviews, observations, surveys and collecting reports.
6. Add the functional and non-functional requirements to your software requirements specification.
7. Using the data you have documented in your SRS in regard to constraints, scope, functional requirements and non-functional requirements:
   a. create a use case diagram for your proposed software solution.
   b. create a context diagram for your proposed software solution.
   c. create a relevant number of data flow diagrams for your proposed software solution.
8. Attach appendices containing any additional information that is required to interpret elements of your SRS.

SCHOOL-ASSESSED TASK TRACKER

☑ Project plan	☑ Justification	☑ Analysis	☐ Folio of alternative designs ideas	☐ Usability tests	☐ Evaluation and assessment	☐ Final submission

9780170440943

CHAPTER

4 Software development: Software design

KEY KNOWLEDGE

After completing this chapter, you will be able to demonstrate knowledge of:

Approaches to problem solving

- techniques for generating design ideas
- criteria for evaluating the alternative design ideas and the efficiency and effectiveness of solutions
- methods of expressing software designs using data dictionaries, mock-ups, object descriptions and pseudocode
- factors influencing the design of solutions, including affordance, interoperability, marketability, security and usability
- characteristics of user experiences, including efficient and effective user interfaces
- development model approaches, including agile, spiral and waterfall.

Interactions and impact

- goals and objectives of organisations and information systems
- key legal requirements relating to the ownership and privacy of data and information.

Reproduced from the VCE Applied Computing Study Design (2020–2023) © VCAA; used with permission.

FOR THE STUDENT

This chapter concludes the discussion of the theory and skills required for Unit 3, Outcome 2.

You will be introduced to generating design ideas, software design, evaluating design options and user experience criteria, as well as privacy and data ownership considerations.

By the end of this chapter, you will be ready to choose one design idea based on your design criteria to further develop in Unit 4, to identify legal requirements and to report your progress.

FOR THE TEACHER

This chapter concludes the theory and skills needed for Unit 3, Outcome 2. Having covered identification and analysis of client and software requirements in chapter 3, students are now introduced to software design, user experience, development model approaches, goals and objectives of information systems, legal requirements and evaluation of alternative design ideas with consideration of efficiency and effectiveness.

By the end of this chapter, students should be equipped to generate several alternative designs and choose a preferred software design according to a evaluation criteria, be ready to use software development tools, and understand user experience characteristics and legal requirements for user data and information.

Note: Students will develop their own software product for the SAT, including identifying the software solution.

iStock.com/Rawpixel

Continuing Unit 3, Outcome 2

In Chapter 3, you learned how to create a software requirements specification (SRS) after consulting widely and analysing the need or opportunity.

The SRS provides the framework for you to consider when designing your solutions. Your design options will address factors identified in the SRS. These factors are all relevant preparation for Unit 3, Outcome 2. For this Outcome, you will fully specify one design that that has been chosen from the several alternative designs you have created and documented.

This chapter begins with a discussion of how to design a software solution and to identify the relevant elements that will lead to a successful project. Next, we will talk about how to determine the characteristics of a positive user experience, both in terms of efficiency and effectiveness of the user interface. We will also talk about approaches by different software development models, types of information system goals and objectives, and important legal requirements for the privacy of **data** and **information** that apply to your Outcome.

FIGURE 4.1 Chapter map

Software solution specifications

In order to achieve a successful software design, the software developer must satisfy all design specifications and parameters. The measure for successful delivery of a software product is a combination of meeting user expectations and requirements as detailed in the SRS, followed by an **evaluation** of the client response to the final product. This is usually assessed through client or user opinion surveys, user interviews and by observing users.

If a feature or function is not included in the specifications, it will not be in the design and it will not be an evaluated design criterion. The designed software solution is created as a response to the SRS. Other factors, if discovered after the SRS has been finalised, can create chaos, if not well documented. Your project management progress report and Gantt chart should include any changes to the SRS and to the **design brief**. Any subsequent changes to the software development process have implications for the evaluation of the final product and must be included.

Generating design ideas

Generating **design ideas** requires a logical approach as well as a creative mind. These thinking skills are referred to as convergent thinking and divergent thinking respectively. There are several aspects to design that need to be considered. The estimated experience of the expected users may be influenced by the previous experience of the software designers and developers. Generally, there is low to very low awareness of accessibility issues when designing software applications. Designing products that can be used by people with a wide range of abilities and disabilities is called **universal design**.

Some individuals use specialised software and hardware called **assistive technology** to operate software products. For example, a person who is blind might use a screen reader program with a speech synthesiser to access the content and functionality of a program.

SCHOOL-ASSESSED TASK TRACKER

☑ Project plan	☑ Justification	☑ Analysis	☑ Folio of alternative design ideas	☐ Usability tests	☐ Evaluation and assessment	☐ Final submission

9780170440943

Convergent thinking

Convergent thinking involves coming up with a single, well-established answer to a problem. All avenues are explored when considering possible solutions, and the best solution is found, ignoring all constraints. This type of thinking involves completing research, such as looking at other software programs that achieve similar purposes, or visiting other companies in the same field to interview or observe their employees. Convergent thinking results in design ideas that are based on other, proven ideas. Data is used from interviews, reports, observations and surveys (see chapter 3), and this data is extrapolated to provide an optimal solution to the problem.

Divergent thinking

Divergent thinking is sometimes described as finding solutions 'outside the box'.

Divergent thinking is more creative than convergent thinking. It involves exploring many possible solutions using spontaneous, free-flowing techniques, such as mind mapping, brainstorming, meditation and role-playing. Divergent thinking involves considering as many possible solutions as you can in a given amount of time. Problems are often explored using stream-of-consciousness techniques. These techniques typically produce unexpected solutions that may not necessarily have been considered using convergent thinking techniques.

Combining thinking skills

On their own, neither convergent thinking nor divergent thinking are likely to produce the most efficient and effective design idea. Convergent thinking ignores constraints that are likely non-negotiable, and divergent thinking may never produce an optimal result. The most effective method of generating design ideas is to use a combination of these thinking skills to produce design ideas that are worth further exploration.

Techniques for generating design ideas

There are several techniques for generating a range of creative and appropriate design ideas. The Study Design does not list specific design techniques that you must know, but the techniques discussed here are the most common techniques. They all aim to find the most effective and efficient software solution to solve a client problem. Your techniques should take into account the functional and non-functional requirements of your solution.

RESEARCH

Functional and non-functional requirements

1. Using the definitions of both terms as set out in chapter 3 as a guide to help you, identify:
   a. the functional requirements of your solution
   b. the non-functional requirements of your solution.
2. Justify your decision.
3. Discuss the functional and non-functional requirements of your solution in class with others.
4. Suggest how a design technique could take into account both functional and non-functional requirements.

9780170440943

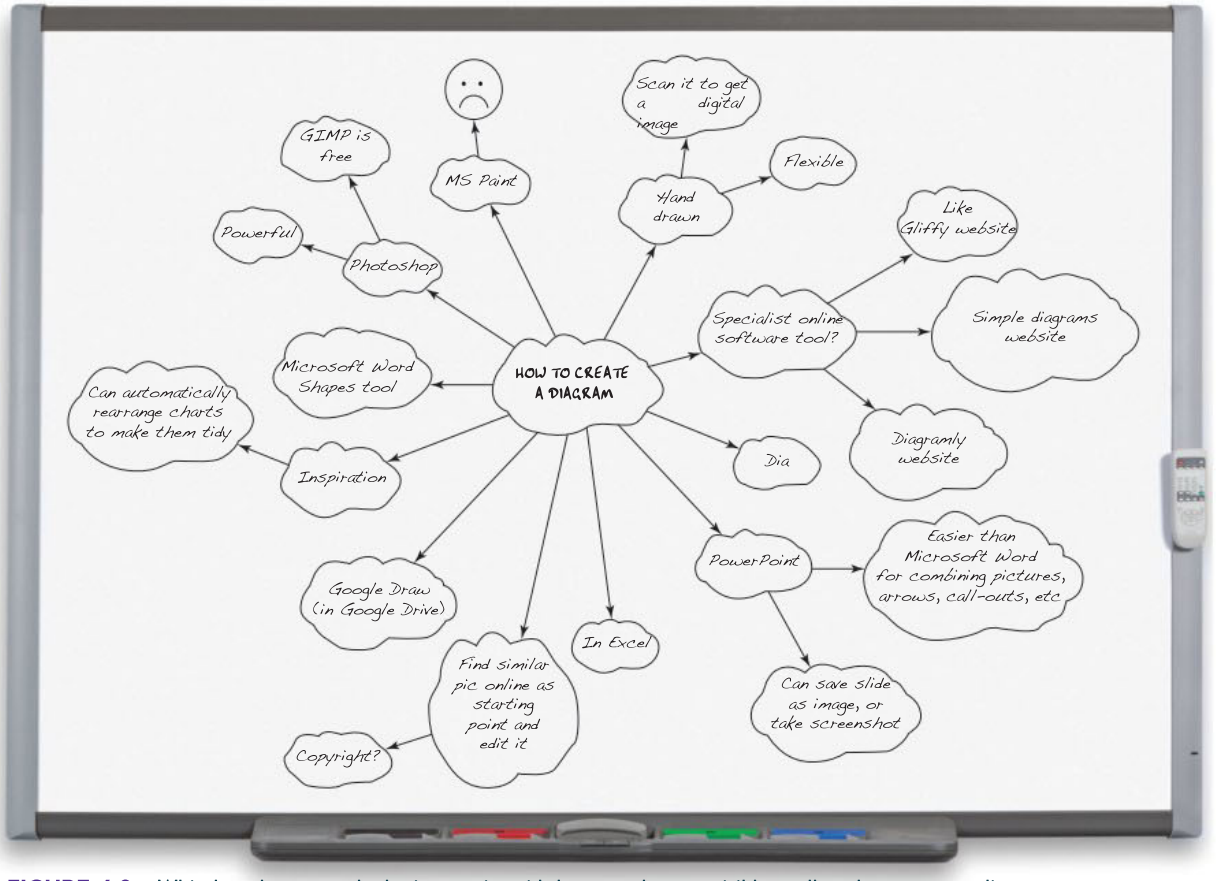
Brainstorming

Brainstorming is a process where ideas are presented in a non-judgemental, spontaneous, unstructured and admittedly somewhat haphazard process. The only rule is that no idea is criticised or rejected; every idea, no matter how outrageous or silly, goes onto a list of possible solutions. Try not to hamper your imagination by rejecting ideas too soon.

Participants must have no fear of being judged, making mistakes or breaking rules. While some ideas may turn out to be ridiculous, sometimes a half-baked, half-comical concept may in fact turn out to be creative genius, or it may stimulate a related idea that turns out to be perfect.

There are certain rules that you need to follow when running a brainstorming session. The most important one is that no one judges any contribution – all suggestions must be accepted. An idea that may seem slightly crazy to begin with can sometimes be workshopped into something great. In the 1970s, a brainstorm produced the idea of a pet rock. The idea was workshopped, and before long you could buy not only a pet rock, but also a pet rock house and a training manual. The idea made the company millions of dollars. It was the pre-technological version of the 1990s Tamagotchi.

Make sure that everyone listens to everyone else's ideas. Make sure that only one person talks at a time, and that there is only one idea at a time. This ensures that even the most shy member of the group will contribute, and also makes it easier to record the ideas. Using these rules will help you gather a large number of ideas to work with.



Shutterstock.com/Rtimages

FIGURE 4.2 Whiteboards are popular brainstorming aids because they are visible to all, and are easy to edit.

SCHOOL-ASSESSED TASK TRACKER

☑ Project plan	☑ Justification	☑ Analysis	☑ Folio of alternative design ideas	☐ Usability tests	☐ Evaluation and assessment	☐ Final submission

9780170440943

Brainstorming example: NASA faces the problem of lifting people and equipment into space. It is extremely difficult, expensive, loud and dangerous. How can it be improved?

Brainstorms for this project include:

- helium balloons: float up, take off
- fire the rockets from the tops of mountains: reduce the distance to space
- a very, very tall ladder
- a giant catapult
- a jet airliner carries the rocket ship as high as it can, then the rocket takes off from there
- antigravity.

While antigravity has no foundation in real science, some of the other design ideas could work, and warrant more research. The team chuckles at the funny 'very, very tall ladder' idea until one person pauses and says:

'Wait … I wonder if we could somehow get a super strong cable from the ground to low Earth orbit and anchor it in space, like a space elevator. You would ride up the cable to the end. The rocket can take off from there. You wouldn't need all the fuel to achieve escape velocity … no need to launch from the ground.'

'And … and re-entry,' says someone else. 'You could ride down the cable to get home. Simple. And low cost.'

From thinking that was whimsical, impromptu, unconventional and unconcerned with constraints came a serious concept that was further investigated by scientists at the Shizuoka University in Japan, with deployment of a prototype in October 2018. They aim to have a fully functional space elevator by 2050.

Brainstorming is helped by including people with different skills, experiences and areas of expertise. Sometimes, a group of specialists struggling for a solution may be inspired by an idea from someone who is not constrained by their shared assumptions, preconceptions and modes of thought.

Consult end users

Your solution, and all information solutions, will be used by real people. Thus, it makes sense to include real people in the design stage, rather than wait for the **testing** and evaluation stages of the problem-solving methodology (PSM) to find out what they think of the solution. Manufacturers, political campaign organisers and film producers are known for their use of 'focus groups' – groups of ordinary consumers who they ask about their likes, dislikes and reactions to design ideas.

A dedicated team of specialist designers may have their own ideas of what an end user wants, but you should value primary evidence of your audience's requirements.

Mind mapping

Mind mapping is an ideal technique to complement the process of brainstorming. Mind mapping involves quickly generating and linking ideas. It is a creative and flexible tool that enables you to add, connect, organise and reorganise ideas (see Figure 4.2). Mind-mapping software is generally flexible enough that you will not need to stop very often to learn how it works while mapping; in other words, your creative flow will not be interrupted.

Unlike whiteboards or physical sheets of butcher's paper, electronic mind maps can stretch endlessly in any direction. It is easy to add or remove links between items, or move

9780170440943

entire branches of thought to new locations. Also, you will not have to copy out all of the scribbled ideas at the end of the session. The mind map can be saved for later development, printed, saved as an image, or transferred to a word processor.

Using the 'getting to space' problem, an example of a mind map of the design process is shown in Figure 4.3.

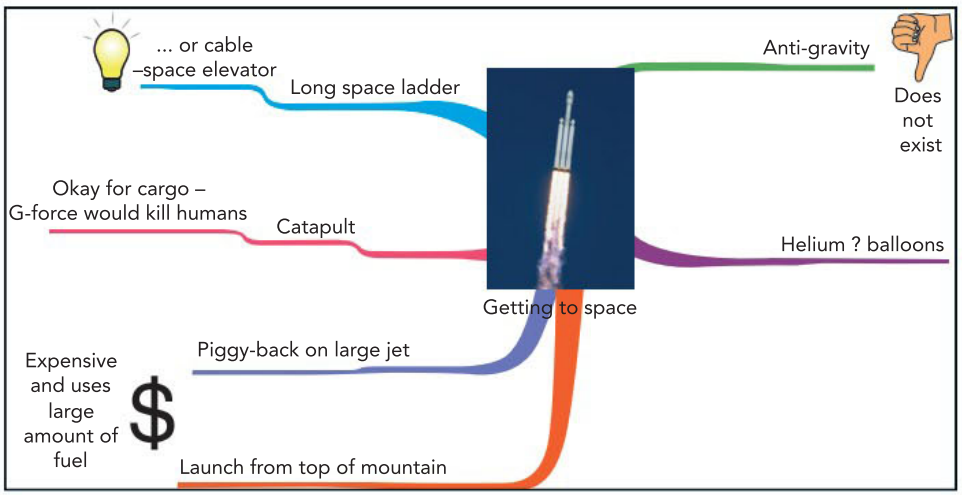


FIGURE 4.3 Mind mapping a project about how to get into space, using Inspiration software

Graphic organisers

Graphic organisers are visual methods of organising ideas. One popular type of graphic organiser is a PMI. A PMI involves organising ideas into three columns: what has been successful (**P**lus), what was unsuccessful (**M**inus) and what needs more thought (**I**nteresting). You can use a PMI to reflect and evaluate, or to brainstorm new ideas (see Table 4.1).

TABLE 4.1 Example of a PMI

Using helium balloons to reach space		
P	**M**	**I**
Quiet	Crashes if gas leaks	Can balloon go high enough?
Relatively cheap	Slow to reach stratosphere	How much does helium cost?
Limited payload weight		

A spider diagram (Figure 4.4) is a powerful tool that gives an overview of a central idea. The body of the spider is the central idea and the branching legs radiate out to related ideas and sub-ideas.

There are dozens of variations of such visual tools to help organise and clarify ideas. Others include character maps, concept webs, POOCh (**P**roblem – **O**ptions – **O**utcomes – **Ch**oice), ranking ladder (to prioritise or rank ideas, information or tasks), stair steps (to organise a process step-by-step), a chain of events, sequence charts (to put sequential factors in order), pie charts (to represent the relative sizes of components in a whole), bone charts, organising trees and even Gantt charts for managing project timelines. Gantt charts were discussed in chapter 3 (see page 77).

9780170440943

Edward deBono became well known for 'lateral thinking'. Mind mapping, rubrics, POOCh and SCAMPER are some of his approaches worth considering.

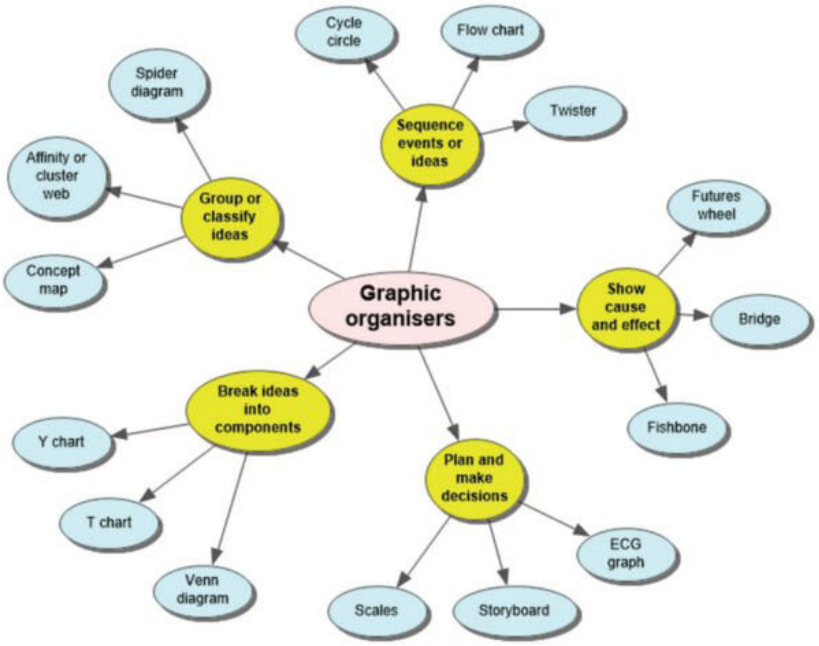


FIGURE 4.4 A spider diagram showing related concepts and sub-classes of concepts; this one was created with Inspiration software. Other suitable software is LucidChart.

Tips for creative thinking

Creative design can be learned. You do not need to be born with the talent. There are techniques that anyone can use to improve their design creativity.

Substitute

Replace part of the problem with something else. For example, if you are producing hundreds of certificates, do not use mail merge to take data from a spreadsheet and insert it into a word processor. Use a database instead.

Getty Images/Photofusion/UIG

FIGURE 4.5 A Raspberry Pi

Combine

Join unconnected things together, such as reducing the weight of camping supplies by combining a spoon and fork into a single utensil – the spork.

Adapt

Use an existing component in a different way. For example, use presentation software such as Powerpoint or Keynote to create a poster. The first spreadsheet was created using the concept of paper-based accounting books.

Strip

Strip the problem right back to its most basic parts and see what is left. For example, the tiny and cheap computer, the Raspberry Pi, is a stripped-down Linux PC with minimal components. Inspecting the basics may reveal the nature of a problem more clearly.

Compare

Ask yourself, 'What other thing do I know that resembles this problem, and how does that other thing work?' For example, when sending a number of print jobs to a single printer, how can they be handled? Like a group of people waiting at a gate, you could organise them into a queue and process them in the order of their arrival.

9780170440943

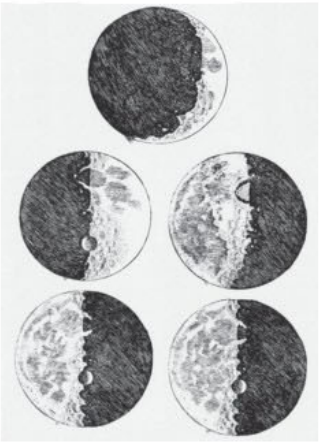
Sleep on it

Creators often reach a point where they can make no further progress. Rather than dwelling on the same failed ideas, it is often better to let them go and think of something else. While the front of your brain is enjoying a wrestling match with your sibling, or an episode of *Australia's Got Talent*, the back of your brain will busily be pulling ideas together to create a solution.

Research

Thomas Edison said, 'Through all the years of experimenting and research, I never once made a discovery. I started where the last person left off.' It is important to learn from your predecessors so you don't waste time 'reinventing the wheel'.

You're unlikely to be the first person to have faced a particular problem. How have other people solved problems similar to the one you face? How have others coped? Their successes may lead you in the right direction, and their failures may prevent you wasting time. Take care when using Google. Make sure you acknowledge your sources, and watch out for false information.



Alamy Stock Photo/Stocktrek Images, Inc

FIGURE 4.6 Galileo's drawings of phases of the Moon, based on observations through his telescope, from his 1610 manuscript, Sidereus Nuncius

Visualisation

Geniuses often represent their thoughts visually because words cannot adequately convey their ideas. Einstein was famous for his non-verbal thought experiments. He visualised travel at the speed of light as travelling on a train. He said that written words and numbers did not play a significant role in his thinking process.

Leonardo da Vinci is renowned for his sketches of his inventions. Galileo Galilei drew diagrams and maps of planetary orbits and phases of the Moon (Figure 4.6). Sigmund Freud, Alfred Hitchcock, Isaac Asimov, Beethoven and Mozart all reported the use of mental imagery in their creative processes. Dr Temple Grandin, famous for her work with livestock, said:

> 'I think in pictures. Words are like a second language to me … Language-based thinkers often find this phenomenon difficult to understand, but in my job as an equipment designer for the livestock industry, visual thinking is a tremendous advantage.'

You may choose to use software simulations or models to help structure your thinking and construct knowledge (Figure 4.7).

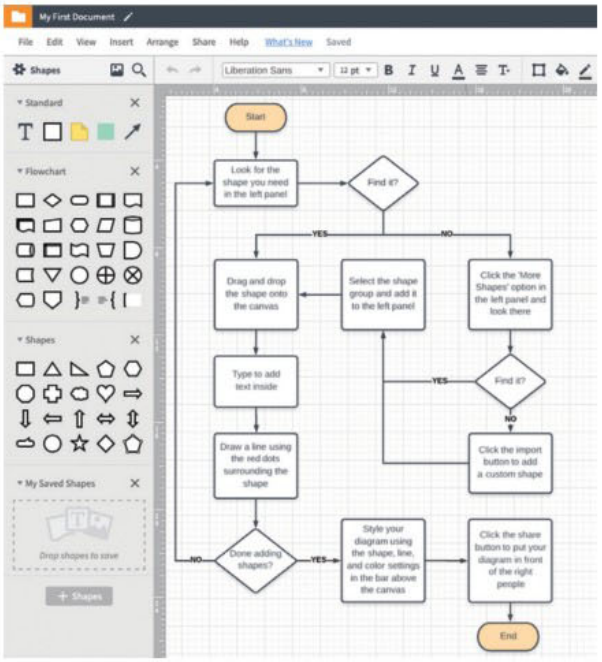


Diagram created in Lucidchart - www.lucidchart.com

FIGURE 4.7 Creating a flow diagram with LucidChart

Be observant and prepared

Many inventions have arisen from people seeing things that were similar to the problem they were trying to solve. Can a blockage in a canal be similar to solving a blockage in blood vessels? How can thousands of ants travel safely and quickly through a small gap, while a crowd of human spectators takes nearly an hour to leave a football stadium?

Play Doh, Post-It Notes, potato chips, Velcro, Teflon, Cellophane, insulin, Dynamite, stainless steel, Super Glue, Cornflakes and vulcanised rubber were all found by observant people after accidents or failed attempts to invent something else.

SCHOOL-ASSESSED TASK TRACKER

☑ Project plan	☑ Justification	☑ Analysis	☑ Folio of alternative design ideas	☐ Usability tests	☐ Evaluation and assessment	☐ Final submission

9780170440943

Research also suggests that creative people are typically hoarders – they keep lots of knick knacks, photos and articles around, and revisit these for stimulus.

Keep your eyes open for connections between apparently dissimilar things. Revolutionary ideas often come from 'ridiculous' connections that no one previously considered. Physicists argued whether light was a wave or a particle, until someone innocently (and correctly) proposed that it could be both.

Someone with a solid knowledge of a topic and an ongoing curiosity about new ideas is receptive, and this will help them recognise the importance of an observation to an existing idea. The uncreative observer will either not notice the idea, or will fail to see its relevance to a developing design.

Take risks, persist and be brave

Someone with a creative design idea often needs to take the risk of being dismissed, mocked or rejected. Many of the greatest breakthroughs were rejected at first, and took a lot of time and effort to be accepted.

The Germ theory, that diseases are caused by microorganisms, was put forward by Louis Pasteur in the 1860s. It superseded the miasma theory, which suggested that a poisonous vapour in the air caused diseases. This theory had endured for several centuries. Pasteur's theory was initially mocked, until further experimentation showed it to be most likely correct.

More recently, Steve Wozniak combined the concepts of a typewriter, a calculator and a display. He was envisioning a whole new technological paradigm: the personal computer. His employer at the time, Hewlett-Packard, rejected Steve's concept five times. This led Wozniak to team up with Steve Jobs, which in turn led to the creation of Apple Computer Inc. The idea of a tablet computing device had been tried by Apple and Microsoft and ended in failure. Steve Jobs tried again when the technology was mature, following the development of the iPod and iPhone, and the resultant iPad was this time successful.

James Dyson (of Dyson vacuum cleaner fame) is believed to have created 5000 prototypes of his vacuum cleaner over five years before he got it right.

These examples show that persistence, not genius, is the greatest contributor to success. Quotes by Thomas Edison, developer of the light bulb, phonograph and electric power:

- 'Genius is one per cent inspiration; 99 per cent perspiration.'
- 'Many of life's failures are people who did not realise how close they were to success when they gave up.'
- 'I have constructed 3000 different theories in connection with the electric light, each one of them reasonable and apparently likely to be true. Yet only in two cases did my experiments prove the truth of my theory.'

Evaluating design ideas

The criteria that should be used to evaluate design ideas must be based on the software requirements specification (SRS) produced at the end of the analysis stage. Elements such as the constraints, functional requirements and non-functional requirements should be considered carefully, and each design should be evaluated in relation to these components before a preferred design is selected. For example, a decision to be made in relation to functional requirements would be the file format in which to save data: plain text, CSV or XML (see pages 17–21). There are advantages and disadvantages to each, which would

9780170440943

then need to be considered against the requirements listed in the SRS before making a final decision. Similarly, the non-functional requirement of usability may result in alternative user interface designs, which must then be evaluated against the needs of the users before a decision is made as to which design is preferred.

When designing the solution to a problem, the first design idea you have will rarely be the best one. A different strategy might be cheaper, easier, faster, more effective, or may better meet the client's needs. While one design idea may be attractive to the developer, the client may have non-technical constraints or priorities that will make another strategy more attractive. Providing a range of design ideas means the client can choose the solution that best suits them.

Although previously proven strategies can be useful, you need to be willing to think outside the box. You may have used a design idea successfully in the past, but it may not be appropriate in the current circumstances. Old strategies will not work for you in every situation.

A successful problem-solver will consider current functional and non-functional requirements and relevant constraints in order to develop an imaginative range of options. The best design idea can then be chosen and developed into a detailed design.

The criteria for choosing the best design idea may include:

- ease of use
- how long it will take to implement
- scalability (how easily the product can be increased in capacity)
- its scope for future modification and enhancement (for example, one design idea may include plug-ins so extra functionality can easily be added)
- the degree to which it satisfies all requirements
- the degree to which it copes with constraints
- ease of implementation
- development cost, future running and maintenance costs
- the amount of labour required to create it and keep it working
- the amount of disruption likely to be caused to the organisation
- compatibility with existing hardware, software, data and procedures
- the amount of training required for staff.

To make an evaluation, the criteria used must ask a question. The answer to that question will indicate whether the criteria has been achieved or satisfied.

For example, some of the previously listed criteria would be evaluated by asking:

- Is the software easy to use?
- Can the software be implemented quickly?
- Can the software be scaled for more users?
- Can the software be simply modified?
- Can the software be implemented easily?

Some design decisions can be very difficult, and require careful balancing of competing needs – usually cost and time against quality. A design that is cheap and quick to produce may quickly wear out, be barely competent or unpleasant to use. A superior design that would lead to a solution with a long life and happy users will probably take longer to produce and cost more.

Compare the likely differences in design philosophies and criteria between the pairs shown in Figure 4.8.

9780170440943

A Formula One race car	A Smart Car
Shutterstock.com/David Acosta Allely	Alamy/Keith Erskine
A PC desktop tower	**An all-in-one Mac**
Courtesy of Dell Inc.	Copyright © 2015 Apple Inc. All rights reserved
A basic, inexpensive coffee table	**An ornate, expensive antique table**
Shutterstock.com/Mariyana M	Shutterstock.com/bergamont

FIGURE 4.8 Pairs

Evaluation criteria should be documented, ideally in a table format, and as each design is evaluated the outcome of each criteria should be recorded.

Testing and evaluation are NOT the same thing! Testing can be done at any time, and may use dummy or test data. Testing looks for specific results and expected behaviour by the known capabilities of programs or equipment. Evaluation is conducted after the software solution has been implemented. Users are consulted and criteria are considered.

Evaluating the efficiency and effectiveness of solutions

Evaluation typically involves checking to see how well a software solution has met its stated requirements. This post-implementation review evaluation is normally performed at a set time period after the solution has been put into place, where the timeframe selected is relevant to the context in which the software operates. However, such an evaluation would not generally occur in the first six months of software being put in place.

9780170440943

In order to conduct an evaluation, an evaluation strategy needs to be created. This normally occurs at the end of the design stage of the problem-solving methodology. The evaluation strategy specifies the timeframe in which an evaluation will occur and outlines the data that will need to be collected to complete the evaluation, including a description of the methods and techniques that will be used to collect that data. It is also made clear how the data collected relates to the criteria that was written in the design stage.

The software solution is then evaluated in terms of **efficiency** and **effectiveness**.

Efficiency

The **efficiency of a solution** concerns how much time, cost and effort has been applied to achieve the intended results. This could include measurements against the speed of processing, the functionality of the software, or the cost of file manipulation.

'Cost' in relation to the efficiency of a solution does not necessarily mean monetary cost. It can also refer to time, as manipulating files can be quite slow depending on the amount of data being manipulated and the selected algorithms that have been implemented to handle that data.

Effectiveness

The **effectiveness of a solution** relates to how well a solution achieves its intended results. This typically requires measurements of the quality of the solution in relation to its **completeness**, **readability**, **attractiveness**, **clarity**, **functionality**, **accuracy**, **accessibility**, **timeliness**, **report formats**, **relevance**, **usability** and **communication of message**.

Some examples of the criteria that could be used to evaluate the effectiveness of a solution are included in Table 4.2.

TABLE 4.2 Criteria for evaluating the effectiveness of a solution

Completeness	Were all of the functional and non-functional requirements that were required by the client implemented in the software system?
Readability	Can every part of the software program be easily read by its users? Are the fonts chosen appropriate in size and face to the system on which the software is installed? Are contrast ratios acceptable? Is the text colour readable against the background colour?
Attractiveness	Are the colours used throughout the software complementary? Are the colour choices appropriate to the context?
Clarity	Is the language used in the software age-appropriate? Are headings, labels and buttons consistently used throughout the software?
Functionality	Does the system respond appropriately to user input errors? What percentage of uptime does the software have?
Accuracy	Is all of the data stored accurate in relation to how it was entered? Are all calculations accurate 100% of the time? Are all reports produced within the correct date ranges, including boundary values?
Accessibility	How well can the system be accessed by someone who is hearing impaired or vision impaired? e.g. Does your solution use a colourblind safe palette? Do all images have an ALT-tag? Will a screen reader work with your solution?
Timeliness	Does the software respond to requests within an acceptable timeframe?
Report formats	Are all of the search/sort reports produced by the system appropriate to their contexts?

9780170440943

TABLE 4.2 Criteria for evaluating the effectiveness of a solution (*continued*)

Relevance	Is all of the information produced and shown by the software system relevant to its intended use? Are there any parts of the system that are not often used (or not at all) by users?
Usability	Are all of the elements within the software easy to use? Are there sections of the program where users are more likely to make errors?
Communication of message	Have appropriate and region-specific conventions been used for all displayed data (e.g. currency conventions, date/time format, alignment)?

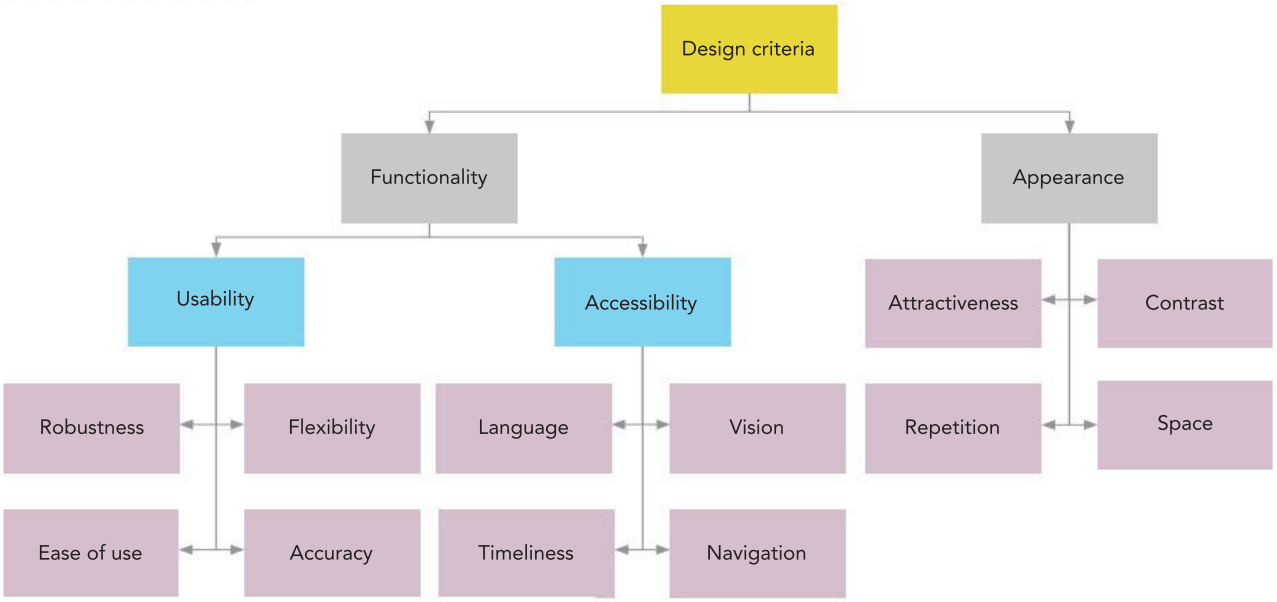


FIGURE 4.9 Design considerations for your software solution

National Center for Accessible Media

FIGURE 4.10 The Web Access Symbol signifies sites or pages where an effort has been made to enable access for disabled users.

If you sense that a page or screen is awkward to use, looks odd or is unattractive, but you cannot exactly say why, you are probably responding intuitively to the use of design in the solution (Figure 4.9).

Methods of expressing software designs

Once you have decided on a design architecture where:

- the use case diagram identifies each entity and accurately shows all relationships
- the context diagram has all aspects of your software solution recorded
- the data flow diagram has been mapped out and your model is consistent

you are ready to begin the software design documentation. Documentation is essential to record definitions, decision details and assumptions that underlie the final software solution. For example, if a date of birth input allowed an unrestricted age range, but a further

calculation assumed only adults would be included, then unexpected results would be generated if an adolescent used the application. This assumption, if documented, would be shown to cause problems in the evaluation stage of the PSM.

There are several methods available to document software designs, including:

- data dictionaries
- object descriptions
- mock-ups (describe appearance)
- wireframes (describe functions)
- pseudocode.

Data dictionaries

There are several types of data dictionary used in the computing industry. In Software Development the data dictionary has a particular meaning, which is different to the definition used in VCE Data Analytics. Both types are included here so you can compare the different purposes.

Data dictionary used as a database design tool

Data dictionaries are used when designing databases to explain how to set up the properties of each field in database tables. Table 4.3 is an example of a data dictionary that can be used in database design.

TABLE 4.3 An example of a data dictionary used by database developers

Modern naming style	Traditional naming style (old school)	Type	Format	Size	Purpose	Example
id	txtCustomerID	Text	XXX99	5	Customer ID	SMI40
firstName	txtFirstName	Text	Xxxxxxxxxxx	15	Customer given name	Jane
lastName	txtFamilyName	Text	Xxxxxxxxxxx	25	Customer family name	Smith
birthDate	dateDOB	Date	YYYY-MM-DD	Fixed	Date of birth	2001-07-19
clubMember	boolClubMember	Boolean	Yes/No	Fixed	Is a member of the buyer's club?	Yes
memberYears	intMemYears	Integer	99	Fixed	Years a member	12
sales	sngSales	Single precision	$##,###.##	Fixed	Total amount spent	$12,456.78

Data dictionary used as a software design tool

A **data dictionary** in software design is used to plan storage structure including **variables**, **arrays** and GUI objects such as text boxes, combo boxes and radio buttons in a program.

It usually lists the names of the variables, their type, size (in characters), scope in regard to the code and purpose of the variable's function. In some cases, it may also list the format and example for the variable.

Developing apps for accessibility – Windows developer

Creating accessible computer applications

Human Interface Guidelines – Apple developer

In 2015 the International Standards Organisation recommended specific ways of naming files and variables: 'singular nouns, present tense verbs, uppercase 1st letter for second and subsequent words. DO NOT prefix names with type or table name'

– ISO 11179-5 :2015

SCHOOL-ASSESSED TASK TRACKER

☑ Project plan	☑ Justification	☑ Analysis	☑ Folio of alternative design ideas	☐ Usability tests	☐ Evaluation and assessment	☐ Final submission

9780170440943

Data dictionaries are most commonly used when code needs to be modified at a later date by programmers. It allows them to understand the purpose of a variable or array that may be unclear without it. Data dictionaries may need to be kept up to date during the development stage, when changes are introduced to the initial design.

TABLE 4.4 An example of a data dictionary used by software developers

Modern naming style	Traditional naming style (old school)	Type	Size	Scope	Format	Purpose	Example
id or ID	`intCustomerID`	Integer	6	Local	999999	Customer ID	201940
firstName	`strFirstName`	Text	50	Local	Xxxxxxxxxxxxxx	Client first name	Jane
lastName	`strLastName`	Text	50	Local	Xxxxxxxxxxxxxx	Client last name	Smith
DOB	`dtDOB`	Date/time	8	Local	YYYY-MM-DD	Client date of birth	2001-07-19
club-Member	`bolClubMember`	Boolean	1	Local	true/false	Is the client a club member?	true
member-Years	`intMemYears`	Integer	2	Local	NN	Loyalty bonus paid to 5, 10, 15 etc. year members	06
sales	`fpSales`	Floating-Point	8	Local	NN,NNN.NN	Total Amount spent	$12,543.76

Differences between the two styles of data dictionary

For VCE Software Development you will only be required to use and answer questions about the software design data dictionary. It is good to know about data dictionaries for database design, but it is not within the scope of this course (it is, however, important for students enrolled in VCE Data Analytics).

The major differences between the data dictionaries are as follows.

- The data dictionary related to software design concentrates on *variables* and *arrays* used in programming, while data dictionaries related to database design focus on data being stored in a database.
- There are a number of field heading differences between each template table design (see Tables 4.3 and 4.4).
- One is used by a computer programmer, the other by a database developer.

Data dictionaries are valuable when code needs to be modified later by other programmers, and the purpose of a variable is not clearly understood.

Note: VCAA Examination questions sometimes use 'old school' variable naming conventions. Be aware of this and be prepared for i. writing your own data dictionary for the design stage of the PSM and ii. responding to examination questions.

Object descriptions

There are many different ways of representing objects that are to be incorporated into the design of the software solution. The basic capabilities for any computer software object are identity, properties and attributes. Object-orientated languages have the added capability of inheritance.

Object descriptions are similar to data dictionaries, but can contain information about methods and events that the object contains. Additionally, object descriptions differ between computer languages.

Figure 4.11 shows an example of ways to define objects. There is not one standard for describing this type of design tool. It is important to understand the purpose and function of each design tool.

Examples of an object description

Example 1:

A class Person has name, date of birth, phone number email address.

A sub-class of Person called Student has enrolled subjects, assessment results.

Another sub-class of Person called Activity has sport, league.

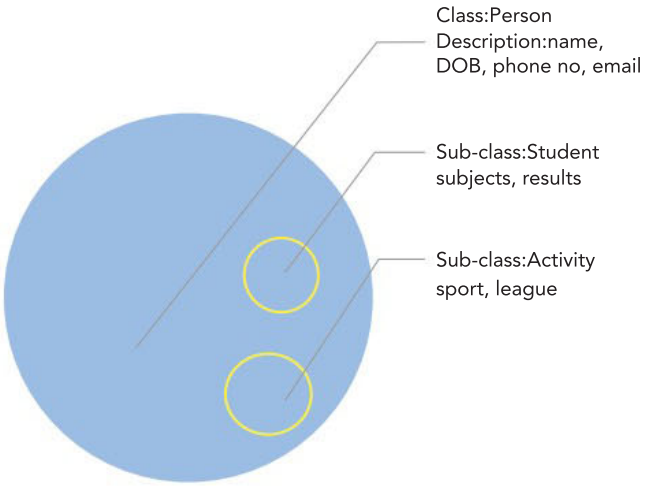


FIGURE 4.11 Example of an object description

Example 2:

Object	Event	Method
Windows	The user clicks on the tick button	Void windowClosing(WindowEvent e)
	The window is opened for the first time	Void windowOpened(WindowEvent e)
	The window is activated	Void windowActivated(WindowEvent e)
	The window is deactivated	Void windowDeactivated(WindowEvent e)
	The window is closed	Void windowClosed(WindowEvent e)
	The window is minimised	Void windowIconified(WindowEvent e)
	The window is maximised	Void windowDeiconified(WindowEvent e)

9780170440943

Example 3:

Object Name: buttonSubmit		
Name	**Type**	**Description**
buttonSubmitClick	Event	When the button is clicked it will submit the transaction and display successful or unsuccessful in a message box on the screen to the user.
buttonSubmitEnabled	Event	Display the button or grey out the button, depending on the user selected options.
buttonSubmitFocus	Method	Change the colour of the Submit button font from black to dark blue when the mouse is hovered over and all fields required to be filled in are valid.

Inheritance is when an object that is derived from an existing base class, a sub-class or 'child', acquires all the properties and behaviours of the super-class or 'parent'.

In some languages (e.g. JAVA), the child can only inherit parent behaviours, NOT grandparent behaviours – i.e. no super-super set of behaviours. C++ can inherit both.

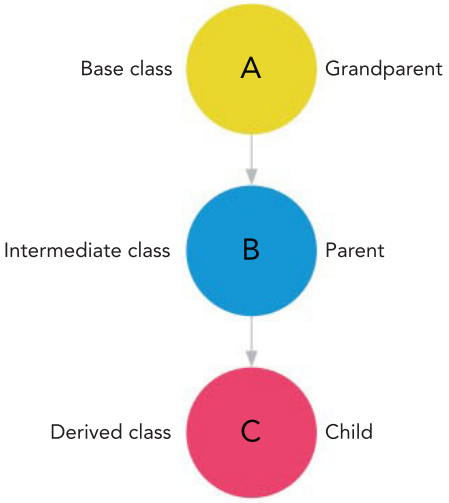


FIGURE 4.12 Multi-level inheritance

Mock-ups

Mock-ups and annotated diagrams

Mock-ups and **annotated** diagrams show the intended appearance of printed output, on-screen information and interfaces.

To design an interface, use a **mock-up**, which is a sketch showing the appearance of the software output. A mock-up should typically include the following features:

- Positions and relative sizes of controls (buttons, scroll bars, status bars)
- Positions, sizes, colours and styles of text (headings and labels, body text)
- Menu positions and contents
- Input boxes, default prompts
- Borders, frames, lines, shapes, images, decoration and colour schemes
- Object alignments (vertical, horizontal, diagonal)
- Contents of headers and footers

Remember, a mock-up can be considered successful if you can give it to another person and they could create the interface without needing to ask you any further questions about it.

9780170440943

Adobe product screenshot reprinted with permission from Adobe

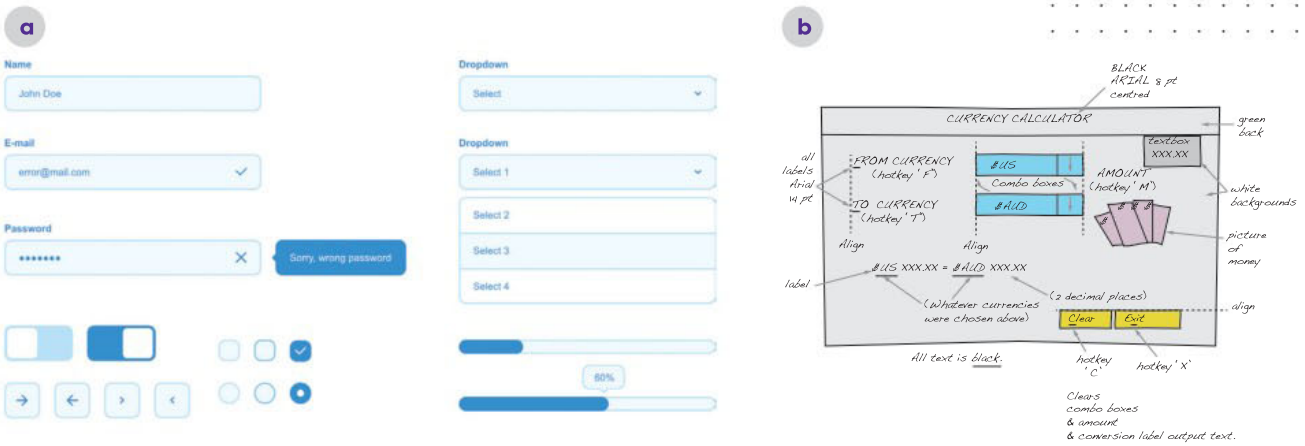


FIGURE 4.13 Input menu options: **a** Mock-up for an app in Adobe XD CC; **b** hand-drawn screen interface

Pseudocode

Pseudocode is intended for human reading rather than machine reading. It appears as informal high-level descriptions of a computer program or other algorithm. A combination of programming terminology and plain English describes algorithms, or instructions, which are easier to understand than programming language code.

Pseudocode describes the logic of the program or algorithm. Each line has one step from the algorithm. Pseudocode is written in structured English and contains control syntax such as:

```
IF-THEN-ELSE a choice is made between two alternatives when a
condition is satisfied (this is also known as boolean)
REPEAT is a loop with a conditional test at the start
FOR-NEXT is a loop with a conditional test at the end
SEQUENCE is when one task is followed by another task
```

Common keywords are used when writing pseudocode. For example:

```
START, END, BEGIN, STOP, DO, WHILE, FOR, UNTIL, REPEAT, IF, THEN,
ELSE, EQUAL, CASE, LESS THAN, GREATER THAN, NOT, OR, TRUE, FALSE,
GET, OPEN, CLOSE, READ, WRITE, END OF FILE, RETURN
```

Combinations of keywords can extend the terms. For example:

```
DO UNTIL, DO WHILE, ELSEIF, END WHILE, END UNTIL, END REPEAT, END IF,
FOREACH NOT EQUAL, NOT OR, XOR
```

Symbols often carry keyword meaning; there are several that have currency. See Table 4.5 for examples. However, there is no single definition of pseudocode.

TABLE 4.5 Common symbols used in pseudocode

Assign value to variable	←
Equivalence (exactly equal to)	= =
Comparison	=, ≠, <, >, ≤, ≥, <=, >=

In VCE Software Development you do not have to use software to create your mock-ups. You may use software if you wish, but you may also create your mock-ups by hand using pen and paper. See the following weblinks for free software tools.

Mockplus
Balsamiq Mockups
(Flash based)
Wireframe CC
Adobe XD CC

SCHOOL-ASSESSED TASK TRACKER

☑ Project plan	☑ Justification	☑ Analysis	☑ Folio of alternative design ideas	☐ Usability tests	☐ Evaluation and assessment	☐ Final submission

9780170440943

Preparation for the VCE examination:
While there are no 'rules' with pseudocode, only conventions, previous VCE examinations have established some specific expectations:
- the assign symbol (left pointing arrow)
- indenting of code.

Other ideas that can be incorporated into your pseudocode include the following.

- Choose sensible variable names.
- Include comments (where necessary, no need to explain the obvious).
- Indent to assist readability and pairing of keywords.
- Ignore unnecessary details.

The purpose of your code needs to be obvious, not too simple, not too difficult. Here are two simple examples of pseudocode.

Compute the area of a rectangle

```
BEGIN
INPUT (or READ) length ← longside,
INPUT (or READ) width ← shortside,
Compute the area ← longside*shortside
PRINT area
END
```

Compute letter grade for assignment score, first three options shown:

```
Enter score ← yourscore
IF (SCORE > 90) THEN
    output A+
    ELSEIF (score > 80) THEN
        output A
        ELSEIF (score > 75) THEN
            output B+
```

Note: there are other ways of achieving this result using ' < ' or using 'CASE'. Further examples of pseudocode were provided in chapter 1, page 16.

Factors influencing solution design

Factors influencing solution design include usability, affordance, security, interoperability and marketability. There are design options that can affect each of these factors. The degree to which each factor is implemented depends on the intended purposes of the client.

There is an interdependence between these factors, so the development of the software product is often a cycle. Each cycle will improve the user experience, or UX, to meet user expectations. The product will be constrained to remain affordable to the target user group.

Security of data and user profiles is an increasingly sensitive topic for software companies and developers. Constant assurance is necessary for continued acceptance by the user community, and any perception that data is gathered for unauthorised purposes will seriously damage the developer's reputation and credibility. Interoperability may become an issue if devices or operating systems require adjustments to guarantee continuity of service. Some updates 'break' the software, requiring massive rewrites. For example, the switch from

Android developers
macOS
winOS
iOS

TABLE 4.6 Timetable showing announcement of 64-bit OS and end of 32-bit applications. At the time of writing, no 32-bit end dates have been notified for desktop/notebook devices.

	64-bit transition announced	64-bit implemented No longer accept 32-bit apps
macOS	2008	tba by Apple
winOS	2009	tba by Microsoft
iOS	2013	October 2015
Android	2017	August 2021

9780170440943

32-bit to 64-bit applications was announced more than 10 years ago for notebooks and desktop devices. Recent OS updates require 64-bit devices and applications. 32-bit applications will no longer operate on some new smartphone devices.

THINK ABOUT SOFTWARE DEVELOPMENT 4.1

Why are smartphone OSs moving to 64-bit applications more quickly than notebook/desktop devices? Identify some factors that have influenced these announcements. For example: The size of the budget for development will be dependent upon the expected return from the sales of the finished software product.

Characteristics of user experiences, including efficient and effective user interfaces

User experience (UX) and **user interface (UI)** may be considered to be the same thing, but there are clear differences and emphases, although effectively they are inseparable. Both processes are oriented to the same ends of getting the content to the user in the most pleasant way.

UX design provides relevant and meaningful experiences for all users. This incorporates hierarchy, navigation and functionality. UI design, on the other hand, has a focus on appearance. These elements include choice of colour, shape, spacing and the look and feel of the software.

What is a successful user interface?

Many experts have studied this issue. If you follow their guidelines you will be on the right track to create a good user interface.

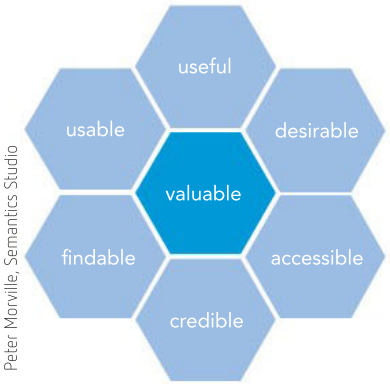


FIGURE 4.14 User experience honeycomb

Developer advice on OS standards
Universal windows platform documentation

However, what is a 'good' user interface or UI? What are the characteristics of a good UI? It should be:

- clear
- concise
- familiar
- responsive
- consistent
- attractive
- efficient
- tolerant.

SCHOOL-ASSESSED TASK TRACKER

☑ Project plan	☑ Justification	☑ Analysis	☑ Folio of alternative design ideas	☐ Usability tests	☐ Evaluation and assessment	☐ Final submission

9780170440943

Used with permission from Microsoft

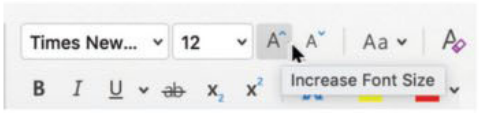


FIGURE 4.15 Clarity: the interface explains what the button will do with a tooltip pop-up to provide an indication of functions.

Copyright © 2019 Apple Inc. All rights reserved.

FIGURE 4.16 Concise: in macOSX the volume controls have tooltip hover and icons to volume levels.

FIGURE 4.17 Familiar icons for save to local storage: HDD, SD card, SSD or USB drive, Save to the cloud, Print, Email, Search.

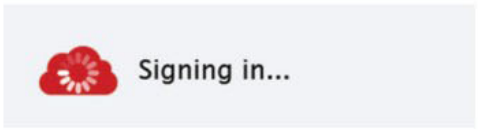


FIGURE 4.18 Wolfram Mathematica has an animated indication icon as the application signs in to Wolfram Cloud.

Clarity

The purpose of the UI is to allow people to interact with your system. Your software solution must communicate meaning and function easily. If people cannot decipher how your application works, or what to do, then confusion and frustration will be the inevitable result (Figure 4.15).

Concise

Clarity is great, but it is easy to fall into the trap of over-clarifying. Be careful that the interface doesn't grow every time you add details. Aim to provide just enough guidance, while still being concise. Use icons wherever possible to reduce the amount of text on the screen (Figure 4.16).

Familiar

The goal for many developers is to create an 'intuitive' interface. This requires the user to recognise and understand how the menu works, as if they have seen it before. Standard icons are easily recognised as buttons with an action (Figure 4.17).

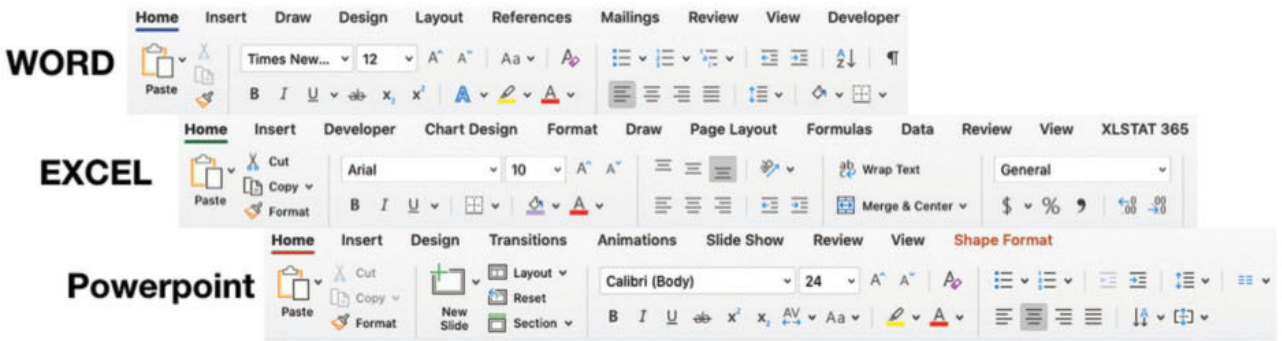
Responsive

There are two possible interpretations of responsive.

1. If your software is responsive, it will operate quickly. There will be no waiting for files to load. An interface that loads quickly will improve the UX.
2. Responsive can also mean the software gives feedback on what is happening, while it is happening. Examples of feedback include: a button animates, changes colour or the text label changes when pressed; a loading bar showing how long the process may take; a spinning icon to show that a process is underway (Figure 4.18).

Consistent

An interface needs to maintain consistency across all screens and pages. A consistent look and feel allows users to develop patterns of use in one context, and to quickly transfer skills and understanding into a new context (FIgure 4.19).



Used with permission from Microsoft

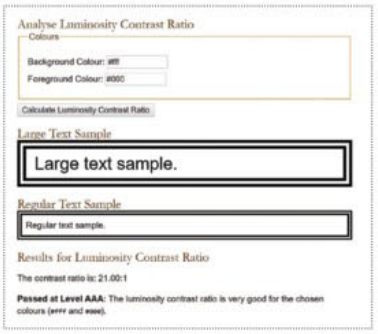
FIGURE 4.19 Microsoft Office menus are consistent for a reason.

9780170440943

Attractive

Appearance is often the first thing observed. Colour, shape and layout all contribute to the appeal of the appearance.

FIGURE 4.20 Colour palette specifications that are colourblind safe



Courtesy of Juicy Studio

FIGURE 4.21 Luminosity Colour Contrast Ratio analyser. A minimum rating of 4.5 to 1 is considered acceptable. A higher value for text and images of text is better.

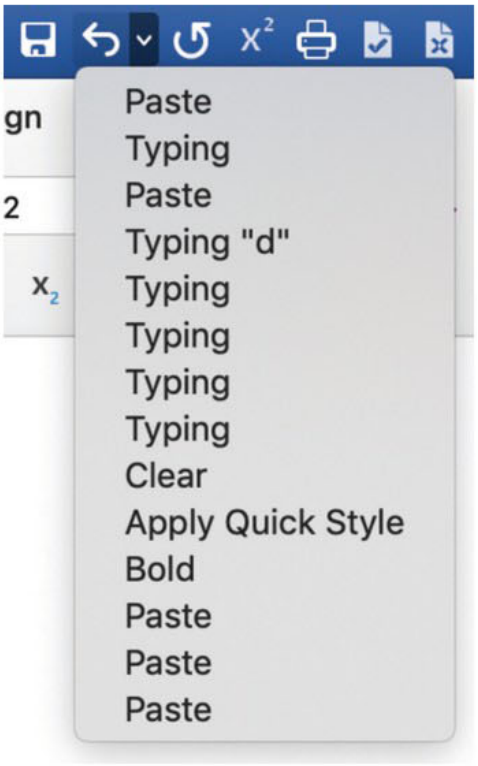
Efficient

Efficiency is the art of providing what the user needs to achieve, with a minimum of fuss. For example, placing frequently chosen items on a screen rather than on a dropdown menu, or on the next screen, gives the user a more satisfying experience, as the task has been achieved without too many clicks.

An efficient interface will achieve a purpose quickly and with minimal effort.

Tolerant

Users make mistakes and change their minds (Figure 4.22). A tolerant interface allows users to retrace their path to choose again. If a file is deleted, can it be recovered? If data is entered incorrectly, can it be changed or deleted simply? If a user chooses a menu item and changes their mind, can they simply 'get back' to the previous screen?



Used with permission from Microsoft

FIGURE 4.22 Microsoft Word has a 20-level undo function.

Design tools are available to assist compliance.

SCHOOL-ASSESSED TASK TRACKER

☑ Project plan	☑ Justification	☑ Analysis	☑ Folio of alternative design ideas	☐ Usability tests	☐ Evaluation and assessment	☐ Final submission

9780170440943

Software development life cycle (SDLC)

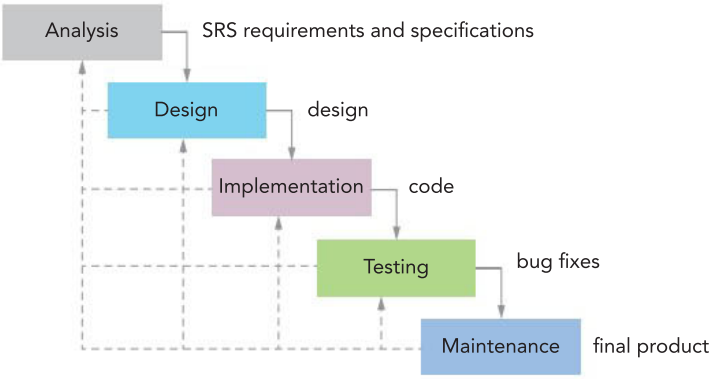
As with all management models and schemes, there are plenty of opinions on how to achieve the 'best' result. As discussed previously, there are many measures of success, so any evaluation criteria need to be carefully considered. Modern development models have been altered to provide the flexibility needed to shift to new information or a change in circumstances. Our Gantt chart is a tool to measure progress against a predicted timeline. We 'know' that timeline will change, so management also needs to change to ensure the project is 'manageable'. The alternative of 'do nothing' would see the project out of control, over time, over budget and under-resourced. In the worst case, it may fail and be abandoned.

Lucy's famous chocolate scene
Which sort of model was used here?

We will explore three **software development life cycle (SDLC)** models in detail: Waterfall, Agile and Spiral. There are several other models. For example, Kanban, rapid prototyping, and scrum are other popular SDLCs.

Waterfall development model

The **Waterfall** model, sometimes described as the 'factory model', requires stages of production to be identified. Each stage must be completed and signed off before the next stage can begin. There is no going back. This kind of production line or conveyor belt approach has the developer working on only one section of the development. Each stage has its own project plan. The Waterfall model was originally proposed by Winston W Royce in 1970 to describe a possible software engineering practice.



Adapted from Bassil, Youssef (2012). A Simulation Model for the Waterfall Software Development Life Cycle. International Journal of Engineering & Technology, vol 2. no. 5.

FIGURE 4.23 The Waterfall software development model

The analysis phase assembles the software requirements specification (SRS). The SRS contains a complete description of the behaviour of the software to be developed, including functional and non-functional requirements.

The design phase plans the software solution, algorithm design, concept design, graphical interface and data structure definitions.

The implementation or development phase constructs the executable program by writing the code, creating the files and the database.

The testing phase verifies and validates that the software meets and satisfies the original specifications.

The maintenance phase occurs after deployment to fine tune output and correct any identified problems or errors, in order to improve performance and reliability. If user circumstances have changed this will be addressed at maintenance/implementation.

TABLE 4.7 Advantages and disadvantages of the Waterfall model

Advantages	Disadvantages
• Simple to use and understand. • Rigid and defined phases allow simple management. • All phases are well documented. • The progress of each phase is visible. • Unlikely to produce unexpected financial expenses. • Testing is simple. • End of project is well defined.	• Rigid process not easily altered. • A long process, with few shortcuts possible. • Detailed progress may be difficult to identify within each stage. • Amplified delays, where a small change in one phase causes delays in all subsequent phases. • Software is available at the end of the project. • Limited opportunities to identify, test and rectify problems or errors.

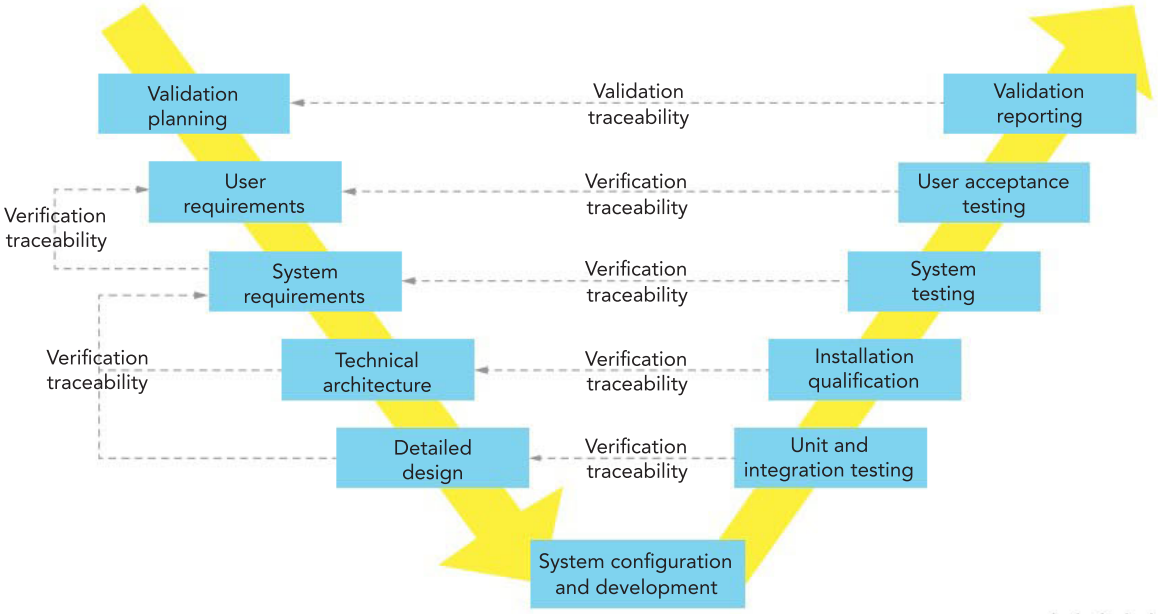
The Waterfall development model would be appropriate for projects:

- that are small in scope
- where the specifications do not change
- where the specifications are well-known, defined and documented
- where there are many dependencies in the project or system.

V-model and the Iterative models

The V-model (also known as the verification and validation model) is a modification of the Waterfall model with testing at each stage, rather than being left to the end. Each stage is completed prior to the next stage beginning, and once a stage is finished it is difficult to go back and make changes.

The Iterative model has short stages that develop functionality. The cycle is repeated until the product satisfies requirements.



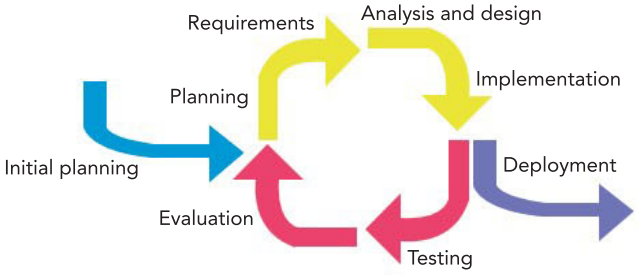
© Courtesy of Testing Excellence, https://www.testingexcellence.com/

FIGURE 4.24 The V-model

SCHOOL-ASSESSED TASK TRACKER

☑ Project plan	☑ Justification	☑ Analysis	☑ Folio of alternative design ideas	☐ Usability tests	☐ Evaluation and assessment	☐ Final submission

9780170440943



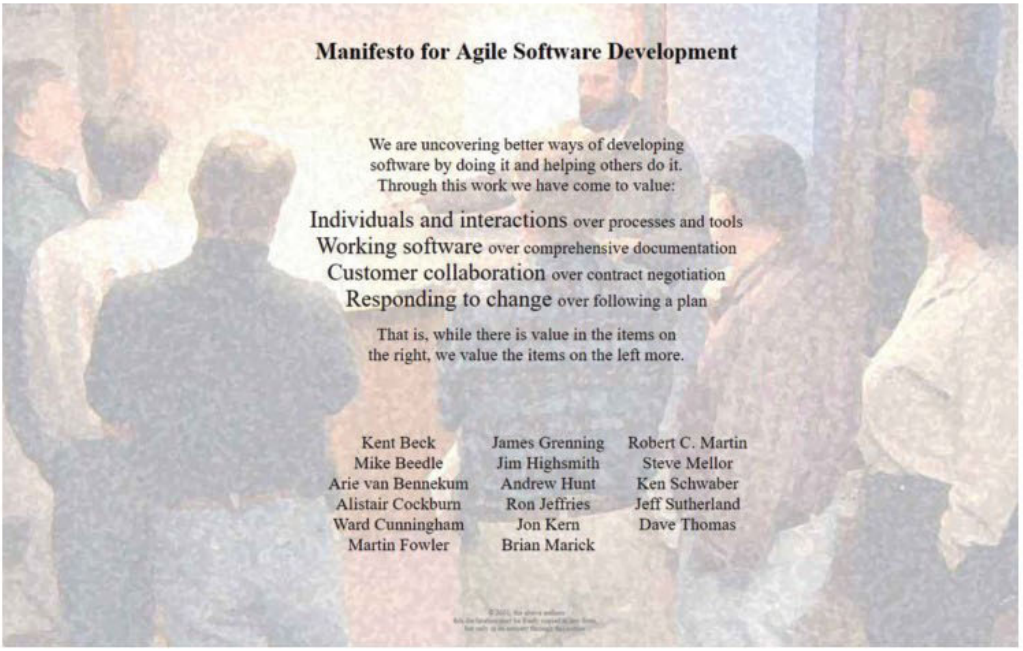
© Courtesy of Testing Excellence, https://www.testingexcellence.com/

FIGURE 4.25 The Iterative model

Agile development model

The Lean and **Agile** models were developed to overcome the deficiencies in plan-driven software development methods such as Waterfall. The Agile model was created in 2001. The four values and 12 principles were written as a response to the heavy management-driven models that were almost universal in the late 20th century and the early 2000s.

The Agile manifesto establishes priorities and stipulates how people working with processes and tools will be supported when working together and delivering value to their clients.



© 2001 http://agilemanifesto.org/

FIGURE 4.26 The Manifesto for Agile Software Development ©2001

The story behind Agile

Individual and interactions	over	Processes and tools
Working software	over	Comprehensive documentation
Customer collaboration	over	Contract negotiation
Responding to change	over	Following a plan

FIGURE 4.27 The Agile methodology

9780170440943

The Agile software development method favours:

- flexibility
- communication
- collaboration
- simplicity.

The key idea for Agile development is iteration. A cycle is repeated, and after each cycle the software shows improvement in a range of areas. The software is under continuous development, and software releases can be more frequent.

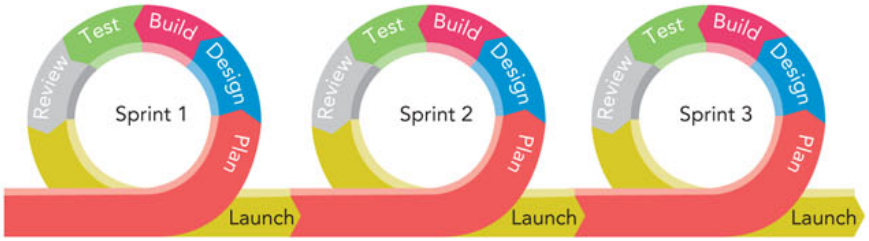
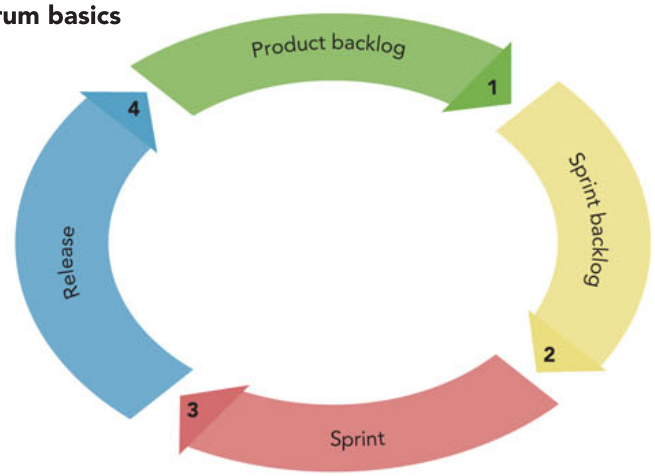


FIGURE 4.28 The Agile software development method is a series of cycles, which continue until the product can be accepted as a **minimum viable product (MVP)**.

Two approaches under the Agile description are Kanban and Scrum. These approaches can also be described as 'lean'. The key difference between Lean and Agile is in the nature of the workflow. Lean is a continuous flow method, whereas Agile begins a new iteration, or cycle, at the completion of each cycle. Most development cycles can be combinations of different schemes, such as 'scrumban'. The emphasis is on the benefits of the method, rather than the name of the definitions.

Scrum basics



Product backlog	Sprint backlog	Sprint	Release
Requirements are arranged and ordered in the product backlog.	The sprint backlog is comprised of the highest priority requirements. The development work to satisfy the requirements becomes one sprint.	A sprint usually lasts between 2 and 4 weeks. A daily sprint refers to the work undertaken in one day.	Each sprint results in key functionality being released or delivered.

FIGURE 4.29 The basics of the Scrum approach

Lean software development originated in 2003.
There are seven principles:

- Eliminate waste
- Amplify learning
- Decide as late as possible
- Deliver as fast as possible
- Empower the team
- Build in integrity
- See the whole

SCHOOL-ASSESSED TASK TRACKER

☑ Project plan	☑ Justification	☑ Analysis	☑ Folio of alternative design ideas	☐ Usability tests	☐ Evaluation and assessment	☐ Final submission

9780170440943

Benefits of the Agile software development cycle

The Agile software development model has short, rapid iteration cycles, called **sprints**. At the end of each sprint, the client can observe and evaluate the result of each iteration and comment on whether the software solution is acceptable. Suggestions can also be made.

A further addition to the weekly cycle is a 'scrum'. A **scrum** is a daily occurrence where developers focus intensely on the issue of the day to resolve as many issues as possible. Overall, the process is unmapped, so the time required and costs can escalate without warning.

TABLE 4.8 Advantages and disadvantages of an Agile model

Advantages	Disadvantages
• Short, rapid interactions segment the project. • Functional requirements are modified quickly. • Risks are reduced. • Early release of first product • Clients are involved closely with developers and testers.	• Developers must be highly skilled, with an ability to listen and respond to clients. • Changes may be inconsistent with previous development. • Timelines may expand beyond expected delivery time.

The Agile software development model would be appropriate where:

- client needs are not well defined or are changing
- the iterations can include many changes at lower cost per implemented modification
- documentation is minimal, as only initial planning is required to begin the project.

Spiral development model

The main benefit of the **Spiral** software development model is repeated iterations of processes and the elimination of risk. Rotation through each sector in turn develops the software solution until a **minimum viable product (MVP)** is achieved. To keep costs down,

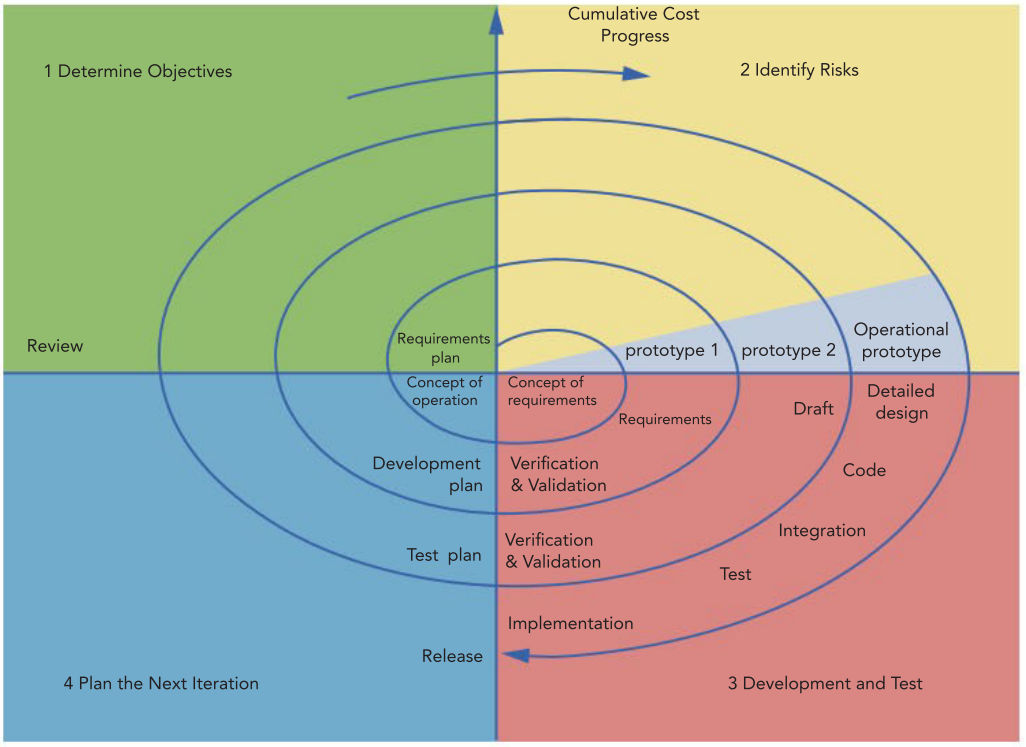


FIGURE 4.30 The spiral model

9780170440943

TABLE 4.9 Advantages and disadvantages of the Spiral model of software development

Advantages	Disadvantages
• Traditional processes remain in place, yet can scale to meet need for more development and testing. • Prototypes are developed quickly; testing begins early. • Risks can be dealt with quickly at each turn of the spiral.	• More expensive than other methods. • More documentation necessary for and between each stage. • An increase in stages for small projects. • High skill sets are necessary for each stage to be effective.

the number of spirals would be kept to a minimum. Each spiral produces a new dot point version of the software solution, which is evaluated. When it reaches a threshold, that product is released and development continues to 'version 2', or the next major 'numeral' version.

Deployment of the Spiral software development model would be appropriate where:

- risks are expected and need to be prevented with quick responsive action
- the client is not certain how to proceed
- products will be released then updated quickly, based on client reactions and feedback
- major edits can be incorporated if developers do not agree.

Goals and objectives of organisations and information systems

Goals help define an organisation's purpose, assist its growth and achieve its financial objectives. Specific goals help measure progress and identify areas for improvement.

Goals need to be specific, measurable, achievable and timely.

There are two types of organisational goals:

- Official: goals that an organisation aims to achieve
- Operative: goals that are necessary to achieve an outcome.

Official goals can be found in public statements and in the organisation's **mission statement**. A mission statement outlines what is important to the organisation.

Organisational goals provide the means by which the organisation's aims can be achieved. Operational goals are short term, can be measured, and enable the organisation to achieve its purpose.

Examples of organisational goals:

- To provide quality products and services
- To develop low-cost, high-value products
- To maintain a well-regarded company reputation
- To provide quality customer service
- To maintain and improve profitability
- To improve client satisfaction.

A goal is usually ambitious and will be achieved over the longer term. Many smaller objectives may be required for the organisation to achieve the larger goal.

Often the nature of a business is reflected in its goals. When important choices need to be made, the goals can provide direction.

THINK ABOUT SOFTWARE DEVELOPMENT 4.2

Use the Internet to locate organisational goals from the mission statements of the following software and digital technology companies.

- Microsoft
- Samsung
- ACER
- Hewlett Packard (HP)
- Huawei

SCHOOL-ASSESSED TASK TRACKER

☑ Project plan	☑ Justification	☑ Analysis	☑ Folio of alternative design ideas	☐ Usability tests	☐ Evaluation and assessment	☐ Final submission

9780170440943

RESEARCH

Match the organisational goal to the company.	
Company	**Organisational goal**
Boeing	To drive prosperity through transport solutions and our vision to be the most desired and successful transport solution provider in the world
Volvo	To refresh the world in mind, body and spirit. To inspire moments of optimism and happiness through our brands and actions
Kia	People working together as a global enterprise for aerospace industry leadership
Volkswagen	To make a contribution to the world by making tools for the mind that advance humankind
Coca-Cola	To offer attractive, safe and environmentally sound vehicles which can compete in an increasingly tough market and set world standards in their respective class
Apple (1977)	To organize the world's information and make it universally accessible and useful
Google	We believe that we are on the face of the earth to make great products ... We are constantly focusing on innovating. We believe in the simple not the complex. We believe that we need to own and control the primary technologies behind the products that we make, and participate only in markets where we can make a significant contribution
Apple (2017)	To become the most successful premium car manufacturer in the car industry
BMW	To blend cultures to become the best and most innovative automotive company in the world by ensuring customer first and mutual prosperity of people

For example: A company has the choice between raising prices or reducing its workforce. If the value is 'good customer experience', then prices will be increased. If the value is 'providing the lowest possible prices', then staffing numbers will be cut.

When values are not upheld, CEOs and directors often resign, as their behaviours and decisions are inconsistent with the company values.

Some common organisational goals

Consider the needs of a simple organisation such as a sporting club. A small club might have only one or two goals. These might be as simple as keeping an accurate record of members' names and addresses and whether they have paid their subscriptions. As the club expands, it is likely that the goals and objectives will grow. Table 4.10 illustrates how the goals and objectives of an organisation can influence the type of information system that needs to be developed.

Every business has a different set of organisational goals. Some have financial goals, such as making large profits, while others want to be more competitive by increasing their market share. The goals will differ depending on the type of organisation.

Other goals that may be included in an organisational mission statement or vision statement include the following.

- Efficiency will reduce waste of time, energy and effort
- Effectiveness will improve how well the operation works
- Governance and decision-making will be clearly communicated
- Establish and maintain reputation

9780170440943

TABLE 4.10 How goals and objectives can influence the type of information system required

Goal	Explanation
Increase the company's profit margin	Businesses exist largely to make money. To provide value to the owners (shareholders or owner/operator), and to allow for further growth and the realisation of opportunities, the business needs to increase its profits.
Expand the company	As businesses want to increase their profit margins, they may find that they need to grow. They may need to employ more people and build larger premises so that their production levels meet customer demand.
Provide quality service	Non-profit organisations such as charities, in particular, would see this as one of their most important goals. They exist to provide service to people who are disadvantaged in the community. A department store such as Myer would also regard this as an important goal, as excellent customer service is paramount to its existence.
Maintain confidentiality	Information stored about customers, products and the workings of a company needs to be protected by an organisation. Organisations need to ensure privacy, and that all information will be treated with confidentiality.

- Improve the client experience
- Customer support
- Supplier support.

Information systems are often created to support the organisational goals. When planning the system, the systems analyst will identify a **system goal**. The system goal explains the specific role of the information system in achieving the organisational goal, and ultimately the company's mission. Setting up the right type of information system can help an organisation make improvements in efficiency, effectiveness and decision-making.

Organisational goals can be assisted by information systems

Information systems usually have specific system goals. They exist to do a particular job, and their success can be measured by specific criteria.

Note the difference between system goals and organisational goals: an organisation may contain many different systems, but the organisation has overall goals it wants to achieve (such as profitability, competitiveness, efficiency, high-quality products, etc.).

These organisational goals should be supported by each system in the organisation, such as the stock control system and payroll system, which have their own specific goals. For example, if an organisational goal is to improve efficiency, each system in the organisation would also need to be efficient.

The goal would set out clear numeric efficiency goals so that both the current situation and the desired goal were quantified. After a period of time gathering data, a measurable improvement (or not) would be obtained.

Some examples of system goals are:

- a payroll system's objective may be to produce accurate employee payroll statements, keep track of tax deductions, and produce summarised statements for management and government departments
- a desktop publishing system's objective may be to produce high-quality page layouts for magazine-quality printing
- a point-of-sale transaction processing system's objective may be to accurately and quickly record purchases, produce customer receipts, and update stock inventories after each sale.

Think of your house as an organisation: its organisational goal is to keep you safe, comfortable and entertained. In the house there are many systems: the doors, the beds and chairs, the TV, radio, computer and bookshelves. The door systems have their system

9780170440943

goals: to prevent unauthorised entry, to keep out insects and unwelcome animals, to keep out rain and wind. The bed/chair systems have their system goals: to provide comfortable support. The entertainment systems' system goals are to provide entertainment. You would not expect a toaster to entertain you because that is not one of its system goals.

Each system in an organisation has a clear purpose and goal. They combine to support the overall goals of the organisation they belong to.

Common system goals are:

- speed
- accuracy
- reliability
- responsiveness
- quality of output
- capacity
- security
- ease of use
- cost effectiveness
- attractiveness of appearance
- flexibility, configuration adaptability and expandability
- safety
- operator comfort
- durability
- robustness, strength, toughness and endurance
- compatibility with other systems.

Legal requirements relating to the ownership and privacy of data and information

Data and information can be very valuable, both to you and to others. Decisions and actions can be made in response to knowing such details, but only if you are authorised to use that information. Unauthorised access to personal and sensitive business information is an increasing problem. Since 1998, in Australia, legislation has been developed to manage information effectively. At the federal level, the *Privacy Act 1988* and, at state level in Victoria, the *Privacy and Data Protection Act 2014* and the *Health Records Act 2001* have been enacted. Periodically, this legislation is updated to reflect the technological advances that impact on information. The *Copyright Act 1968* (which includes the *Copyright Amendment (Digital Agenda) Act 2000*) also limits who can use certain information.

Information is the most valuable asset that an organisation owns. In many cases, the time and resources that have been used to collect the data and assemble the information could not be replicated. If the data and information were lost or damaged, it may not be recoverable. An information systems manager would plan to prevent loss from occurring.

There are several key laws relating to the information systems and telecommunications industries. At a federal level, the law concerned with how information about people can be used is the *Privacy Act 1988*. In Victoria, we are especially concerned with the *Privacy and Data Protection Act 2014* and the *Health Records Act 2001*. Combined, these laws govern

the collection and use of private information by both government and non-government organisations at both state and federal levels. Employers and government agencies have a legal responsibility to ensure that these laws are implemented within their organisations. In addition, organisations must make employees and customers aware of their rights, as well as their responsibilities, in relation to these laws.

Privacy Act 1988

In the mid-1980s, the Federal Government attempted to introduce an 'Australia Card'. This proposal was met with overwhelming resistance from the public, and eventually dropped. In its place, the pre-existing tax file number (TFN) system was updated. There were many concerns about how the Federal Government might use tax file numbers, especially regarding the release of confidential tax information, or matching data from different government departments. Under international law, the government was required to provide adequate protection for personal data being sent to other countries, and to ensure that civil rights with regard to privacy were not being ignored.

Originally, the *Privacy Act 1988* only dealt with the handling of data by government agencies. Many people criticised these limitations because it seemed that private organisations were not required to apply even the most basic of safeguards regarding the data they collected. Even worse, there were no regulations preventing non-government organisations from collecting data by any method and using it for any purpose without consent. The rapid growth of electronic transactions, especially over the Internet, led many people to demand some sort of legal protection from those who might gather data about their browsing habits. The government was keen to encourage the development of electronic commerce, while protecting the confidentiality of consumers and increasing public confidence in electronic transactions. As a result, several amendments were incorporated into the *Privacy Act 1988*. These were the most significant changes to have been made to privacy laws since the inception of the legislation.

There have been several additional powers included within this Act since 1988, but its main purpose has remained unchanged. The *Privacy Act 1988* was amended by the Privacy Amendment (Enhancing Privacy Protection) Bill 2012, which came into effect on 12 March 2014.

'These are the most significant changes to privacy laws in over 25 years and affect a large section of the community. The world has changed remarkably since the late 1980s when the *Privacy Act* was first introduced, and so the changes were required to bring our laws up-to-date with contemporary information handling practices, including global data flows,' said then Australian Privacy Commissioner Timothy Pilgrim in 2014 when the Privacy Amendment (Enhancing Privacy Protection) Bill came into effect.
– Australian Government, Office of the Australian Information Commissioner

Application of the *Privacy Act*

The *Privacy Act* applies to both electronic and manual or conventional forms of data gathering and handling by private organisations. The Act also has provisions specifically addressing the use of personal data for direct marketing via email, which can only be used with the consent of the individual concerned. It also extends to general privacy issues regarding workplace email. The Act encompasses businesses with an annual turnover of $3 million, all private health services that store health records, businesses that trade in personal information, and those organisations that choose to opt-in.

Individuals also have rights under the Act, which makes for provisions on how their personal information is collected. The Act defines personal information as being:

- information or an opinion about an identified individual, or an individual who is reasonably identifiable: whether the information or opinion is true or not; and whether the information or opinion is recorded in a material form or not.

9780170440943

THINK ABOUT SOFTWARE DEVELOPMENT 4.3

Many organisations have a privacy policy listed on their website. Find out what is covered by your school's privacy policy. What information might the school have about students that should not be made publicly accessible?

The amended Act defines personal information as including an individual's:

- name and address
- signature
- telephone number
- date of birth
- medical records and health information
- bank account details
- photos and videos
- biometric and genetic information
- philosophical beliefs
- likes and dislikes
- opinions or commentary about a person
- racial or ethnic origin
- memberships of political associations
- professional or trade associations or trade unions
- religious beliefs or affiliations
- criminal record
- sexual orientation or practices.

Since the introduction of the updated *Privacy Act 1988*, organisations have had to review the way they handle customer information and had to update their technologies and their security processes to ensure they comply with the new legislation.

The Act prescribes severe penalties for serious and repeated interferences with privacy, which can result in criminal prosecution and/or fines of up to $340 000 for individuals and $1 700 000 for public and private organisations.

The amount specified in the *Privacy Act* is not variable. That is, the fine of $340 000 for individuals and $1 700 000 for public and private organisations does not change. The amount will only change if the Act is changed. In Victoria, however, the fines are variable and tied to penalty units with an amount that is adjusted on 1 July each year.

Privacy and Data Protection Act 2014

The *Privacy and Data Protection Act 2014* (PDPA) was introduced by the Victorian Government. It replaced the *Information Privacy Act 2000* and the *Commissioner for Law Enforcement Security Act 2005*. The PDPA is intended to strengthen the protection of personal information and other data held by Victorian government agencies, including local councils and contractors working for the State Government.

Under the PDPA there is a single privacy and data protection framework. The PDPA uses its own Information Privacy Principles (IPPs), and organisations are obliged to act in accordance with the IPPs. As a result of the PDPA, a Privacy and Data Protection Commissioner has been established.

Information Privacy Principles

As discussed in the previous section, the amendments to the *Privacy Act 1988* in 2014 introduced new Australian Privacy Principles (APPs). It was anticipated that the current Victorian Information Privacy Principles (IPPs) would be replaced with new principles based on the Australian Privacy Principles (APPs). However, this has not happened, so the *Victorian Privacy and Data Protection Act 2014* continues to use the IPPs.

APPs are discussed further on pages 267–68.

9780170440943

TABLE 4.11 The 10 Information Privacy Principles (IPPs)

IPP	Description
Collection of personal information	When an organisation collects information, it should only collect the information it needs. The organisation should inform people that their information is being collected.
Use and disclosure of personal information	When an organisation uses and discloses personal information it is only for the purpose that it was collected for, or for a secondary purpose that you would reasonably expect.
Data quality	Ensure that the information collected is accurate, complete and up-to-date.
Data security	Information must be protected from misuse, loss, unauthorised access, modification or disclosure. Reasonable steps must be taken to destroy or de-identify personal information that is no longer needed.
Openness	The organisation needs to be transparent about what it does with information. Non-compliance will result in a maximum penalty for a body corporate of 3000 penalty units and 600 penalty units for an individual.
Access and correction	When an organisation collects information it should allow people to see the information it collects about them and provide them with the opportunity to correct it if it is inaccurate.
Unique identifiers	Use of unique identifiers, usually a number, is only allowed where an organisation can demonstrate that the assignment is necessary to carry out its functions efficiently.
Anonymity	Where possible, people supplying information should be given the option of not identifying themselves.
Transborder data flows	If your personal information travels outside Victoria, your privacy protections must travel with it.
Sensitive information	Organisations need to ensure that they do not collect sensitive information about people, such as their religion, political views or criminal record, without checking the applicable laws.

The new APPs replaced the two sets of principles that have applied to Commonwealth public sector and private sector organisations since 2001. They were known as the Commonwealth Information Privacy Principles and the National Privacy Principles.

Health Records Act 2001

The *Victorian Health Records Act 2001* was created to provide direction regarding the collection and handling of health information in both the public and private sector. It is anticipated that patients will use both private and public health services at various stages of their life. The *Health Records Act* allows people to access their own medical information, as well as establishing the health record privacy principles for both public and private medical services. The *Health Records Act* established 11 Health Privacy Principles to provide rights to both living and deceased people. These principles apply to the collection, use and storage of personal health information in Victoria.

Penalty units define the amount that needs to be paid for offences in Victoria. Generally, the legislation does not specify the monetary amount, but does specify the penalty unit. Each year, the penalty unit is specified. For example, from 1 July 2015 to 30 June 2016, one penalty unit was worth $151.67. The rate for penalty units is indexed each financial year so that it is raised in line with inflation. Changes to the value of a penalty unit take effect on 1 July each year.

THINK ABOUT SOFTWARE DEVELOPMENT 4.4

Stevie is a student support officer at a Victorian government school. He has access to student and parent personal details. He has been approached by an external organisation to 'sell' these details in exchange for new computer equipment for the school.

1 Identify key legislation that Stevie should consider before providing the information to the external company.
2 What are Stevie's ethical responsibilities to the students, parents and the school?

SCHOOL-ASSESSED TASK TRACKER

☑ Project plan	☑ Justification	☑ Analysis	☑ Folio of alternative design ideas	☐ Usability tests	☐ Evaluation and assessment	☐ Final submission

9780170440943

The *Health Records Act 2001* applies to a deceased individual who has been dead for 30 years or less.

From the age of 16, teenagers can consent to medical and dental treatment with the same authority as an adult. Therefore, teenagers can see a doctor by themselves without their parents. At 18 years of age, they have the legal capacity to give consent to, and refuse, medical treatment.

The Act protects the confidentiality of patients' health care information by allowing the information to be used only for the primary purpose for which it was gathered. This means that information about medical test results and your medical history may be used by your doctor, the hospital and any other health professionals only for the purpose of your immediate or ongoing care. This information would not be disclosed to a third party for a 'secondary' purpose (for example, to your medical insurance company or another hospital) without your consent. Health information may, however, be provided to third parties without your consent under certain, and strictly limited, circumstances, including requests by family members in an emergency when you cannot give your consent and your life is threatened, where there is a serious threat to public health and welfare, research in the public interest, investigation of unlawful activity and as part of a legal claim.

An individual who believes that the *Health Records Act* has been breached can make a complaint to the Health Services Commissioner, who will try to achieve a resolution by discussion between the parties. If a satisfactory resolution cannot be reached, the Commissioner may then serve a compliance notice on the organisation that has breached the Act. This notice informs the organisation which area of the Act has been breached, and states that it must correct its procedures. The maximum penalty for an organisation is currently 3000 penalty units, and 600 penalty units for non-corporate cases.

Health Privacy Principles

Table 4.12 presents a summary version of the Health Privacy Principles. This is not the full set or form of the Principles, and is intended for quick reference only. The principles in full can be found in the Act.

TABLE 4.12 A summary of the Health Privacy Principles

Health Privacy Principles summary
1 Collection
Only collect health information if necessary for the performance of a function or activity and with consent (or if it falls within HPP1). Notify individuals about what you do with the information, and that they can gain access to it.
2 Use and disclosure
Only use or disclose health information for the primary purpose for which it was collected or a directly related secondary purpose that the person would reasonably expect. Otherwise, you generally need consent.
3 Data quality
Take reasonable steps to ensure health information you hold is accurate, complete, up-to-date and relevant to the functions you perform.
4 Data security and retention
Safeguard the health information you hold against misuse, loss, unauthorised access and modification. Only destroy or delete health information in accordance with HPP4.
5 Openness
Document clearly expressed policies on your management of health information and make this statement available to anyone who asks for it.

9780170440943

Health Privacy Principles summary
6 Access and correction
Individuals have a right to seek access to health information held about them in the private sector, and to correct it if it is inaccurate, incomplete, misleading or not up to date.*
7 Identifiers
Only assign a number to identify a person if the assignment is reasonably necessary to carry out your functions efficiently.
8 Anonymity
Give individuals the option of not identifying themselves when entering transactions with organisations where this is lawful and practicable.
9 Transborder data flows
Only transfer health information outside Victoria if the organisation receiving it is subject to laws substantially similar to the HPPs.
10 Transfer/closure of practice health service provider
If you are a health service provider, and your business or practice is being sold, transferred or closed down, without you continuing to provide services, you must give notice of the transfer or closure to past service users.
11 Making information available to another health service provider
If you are a health service provider, you must make health information relating to an individual available to another health service provider if requested by the individual.

*In the public sector, individuals already have this right under Freedom of Information.

Department of Health and Human Services, Victoria

Ethics are the principles of right and wrong that are accepted by an individual or a social group. Ethical behaviour often guides policy makers within organisations. For example, a network policy within an organisation governs both the legal and ethical use of the information system.

SCHOOL-ASSESSED TASK TRACKER

☑ Project plan	☑ Justification	☑ Analysis	☑ Folio of alternative design ideas	☐ Usability tests	☐ Evaluation and assessment	☐ Final submission

9780170440943

4 CHAPTER SUMMARY

Essential terms

accessibility how easily the software can be used by those who experience disabilities

accuracy the absence of mistakes or errors

Agile a method of software development that emphasises flexibility and interactive improvement. Developers, testers and clients communicate directly without management involvement.

annotate add comments to a document or diagram

array a collection of values or variables that are sometimes indexed

assistive technology any device or system that is designed for individuals who would otherwise find the task difficult or impossible

attractiveness how pleasing something is to the viewer

clarity ease of understanding

communication of message the process through which meaning is transferred

completeness nothing left out

convergent thinking involves coming up with a single, well-established answer to a problem

data raw, unorganised facts, figures and symbols fed to a computer during the input process; data can also refer to ideas or concepts before they have been refined

data dictionary (software design) used to plan storage structure; provides specifications of variables, arrays and GUI objects

design brief a document or statement that outlines the nature of a problem, opportunity or need

design idea brief, rough strategies for solving a problem; the best design idea will satisfy all functional and non-functional requirements, work within the defined constraints and be efficient

divergent thinking involves exploring many possible solutions using spontaneous, free-flowing techniques

effectiveness produces the expected result

effectiveness of a solution how well the software works to produce the desired result

efficiency economic use of resources with minimum waste

efficiency of a solution whether the result is produced quickly and simply

evaluation an assessment of whether a solution achieves the goals for which it was originally designed; not the same as testing

evaluation criteria rules set out during design that include effectiveness and efficiency criteria; based on the solution's requirements, which were defined during analysis

functionality what the software can do

goal an anticipated result or aim, which is specific, measurable, achievable and timely

information knowledge about a person, place, event or thing

Lean software development a methodology that optimises efficiency and minimises waste

minimum viable product (MVP) Agile methods develop simple prototypes quickly, which meet client software requirements specifications (SRS)

mission statement statement setting out an organisation's purpose or what it is trying to achieve; the mission of most companies is to make a profit, while non-profit organisations tend to define their key mission as providing a service to their members

9780170440943

mock-up a sketch showing the look of the software output

objectives small, achievable tasks undertaken to accomplish a larger task

organisational goal how an organisation intends to go about achieving its mission

pseudocode code that designs algorithms in a clear, human-readable, language-independent format

readability the ease of understanding the text

relevance appropriate meaning in the context of the discussion

report formats the resultant screen after a search or sort

scrum part of the Agile software development process

software development life cycle (SDLC) a plan to develop, test, evaluate and implement a software solution

Spiral a method of software development that adopts an iterative cycle overlaid on the Waterfall method

sprint a method of software development where each interaction cycle is 2–4 weeks, resulting in quicker development of a finished minimum viable product

system goal how the specific role of an information system will help in achieving an organisational goal

timeliness occurring at the right time

testing checking that occurs during development to ensure that a solution works, answers are correct, links resolve, site does not crash and all is fully functional

universal design designing products that can be used by people with a wide range of abilities and disabilities

usability ease of use to achieve specified goals in terms of efficiency, effectiveness and satisfaction

user experience (UX) an emotional reaction when using a device or software

user interface (UI) interaction between the user and the device

variables used in an algorithm or equation to carry values for calculations

Waterfall a method of software development that is lock-step, sequential and traditional. Once a stage is completed, there is no going back.

Important facts

1. A **detailed design** is produced based on a design idea that best solves the problem.
2. **Design ideas** may come from **brainstorming**, using outside consultants, talking to end users, **mind mapping, graphic organisers, attribute listing**, research and **looking at parts of the problem in different ways**.
3. Gantt charts and project plans should be updated on an ongoing basis to reflect real events as projects proceed.
4. Designing products that can be used by people with a wide range of abilities and disabilities is called '**universaldesign**'.
5. Include **real people** in the design stage, rather than waiting for the testing and evaluation stages.
6. Criteria used to evaluate design ideas must be based on the **software requirements specification (SRS)**.
7. Testing and evaluation are NOT the same thing.
8. The **efficiency** of a solution concerns how much time, cost and effort has been applied to achieve the intended results.
9. The **effectiveness** of a solution relates to how well a solution achieves its intended results.
10. **Documentation** is essential to record definitions, decision details and assumptions that underlie the final software solution.
11. **Data dictionary (programming)** specifies details of variables, arrays and objects.
12. Security of data and user profiles is an increasingly important topic for software companies and developers.
13. **UX design** provides a relevant and meaningful experience for all users.
14. **UI design** has a focus on appearance.

9780170440943

4 CHAPTER SUMMARY

15 The **Waterfall software development model**, often described as the 'factory model', has identified stages of production. Once each stage is completed, it is 'signed off' so the next stage can begin. There is no going back.

16 The **Agile software development method** achieves flexibility, communication, collaboration and simplicity through iteration.

17 Agile has short, rapid iteration cycles called **'sprints'**. At the end of each sprint a working model is produced and the client has an opportunity to observe, evaluate and comment on whether the product is acceptable.

18 The **Spiral software development method** has repeated iterations of processes and aims to eliminate risk.

19 **Goals** help define an organisation's purpose, assist its growth and achieve its financial objectives.

20 Goals need to be **specific, measurable, achievable** and **timely**.

21 A **mission statement** outlines what is important to the organisation. **Organisational goals** provide the means by which the organisation's aims can be achieved.

22 **Information systems** usually have specific **system goals**. They exist to do a particular job, and their success can be measured by specific criteria.

23 Unauthorised access to personal and sensitive business information is an increasing problem.

24 **Information** is obtained when **data is manipulated** by the computer's processor into a meaningful and useful form (also becoming its output).

25 At a federal level, the law concerned with how information about people can be used is the *Privacy Act 1988*.

26 As part of the *Privacy Act*, the Australian Privacy Principles (APPs) were devised to set out the standards, rights and obligations for collecting, handling, holding, accessing, using, disclosing and correcting personal information.

27 The *Privacy and Data Protection Act 2014* (PDPA) was introduced by the Victorian Government. It replaced the *Information Privacy Act 2000* and the *Commissioner for Law Enforcement Security Act 2005*.

28 The other key Victorian law relating to privacy is the *Health Records Act 2001*. This Act governs the collection and handling of confidential medical records.

9780170440943

TEST YOUR KNOWLEDGE

Review quiz

Generating design ideas

1 What is the difference between convergent thinking and divergent thinking?
2 What is the most important rule to follow when you are conducting a brainstorming session?
3 Why is it important to show your solution to the end user?
4 Explain how you would use mind mapping to follow up a brainstorming session.
5 Provide one example of an historical breakthrough. What technique was used to get to that breakthrough?

Evaluating efficiency and effectiveness

6 What is the difference between effectiveness and efficiency?
7 Explain how an improvement in efficiency may cause a decrease in effectiveness.
8 Describe how efficiency and effectiveness might be measured. How might these quantities be measured?
9 The statement, 'Testing is not evaluation' suggests that evaluation is not about test tables and input–output expected results. Identify some of the quantities that evaluation will measure.
10 When should evaluation take place?

Software development life cycle

11 Why can the Waterfall method be described as a linear model?
12 What is the Agile methodology when it is applied to software development?
13 Describe any similarities and differences between the Waterfall software development method and an Agile SDLC method.

Goals and objectives of organisations and information systems

14 Explain the difference between an organisational goal and a mission statement.
15 What is a vision statement?
16 How are values important to the content of mission and vision statements?
17 Where does the purpose of mission and vision overlap?
18 Why are mission and vision important for organisational goals and objectives?

SCHOOL-ASSESSED TASK TRACKER

☑ Project plan	☑ Justification	☑ Analysis	☑ Folio of alternative design ideas	☐ Usability tests	☐ Evaluation and assessment	☐ Final submission

9780170440943

TEST YOUR KNOWLEDGE

19 Apart from making a profit, think of an organisational goal for each of the following types of organisations:

- **a** A veterinary clinic
- **b** A large shopping centre
- **c** An airline
- **d** A brewery
- **e** A public library

Information management strategies

20 Explain why an organisation must comply with legal requirements.

21 Briefly summarise the role and scope of the three key laws affecting privacy of information.

22 Why have these laws been introduced?

23 If you believe that the privacy of your information has been breached by the Australian Taxation Office, to whom can you complain?

24 What are the penalties for breaches of the *Privacy and Data Protection Act 2014*?

25 For each of the following breaches of privacy, suggest which privacy law would apply.

- **a** You find that your employer has published your tax file number on the Internet.
- **b** Medical records are found at the tip.
- **c** A bank refuses to give you a loan because the manager claims your credit record is poor, when it should actually be very good.
- **d** A consultant working for the Victorian Government passes on your VCE results to a friend without your permission.
- **e** A website you visit asks for personal information from you, but does not display its privacy policy.

9780170440943

APPLY YOUR KNOWLEDGE

1 Match the evaluation criteria with the method of evaluation. Identify which criterion is efficiency and which is effectiveness.

Evaluation criteria	Method of evaluation	
The system should have over 95% availability.	Opinion interviews of 500 users at random	**e**
User survey should achieve a very high satisfaction level.	Operators record time on task and other duties	**f**
20% less operator time with adoption of the new system	Online survey of users	**g**
Usability of the system should be rated highly	A log of faults records the time and duration of system failure.	**h**

2 A large Australian financial institution sends a policy document to all clients on how the company handles personal information. The document includes the following statements:

> ### Your personal information
>
> Personal information is information or opinions that allows others to identify you. It includes your name, age, gender, contact details, as well as your health and financial information. We will act to protect your personal information in accordance with the National Privacy Principles or an industry privacy code. We collect personal information to provide you with the products and services you request as well as information on other products and services offered by or through us.
>
> ### How we use your personal information
>
> Personal information may be used and disclosed within the company to administer our products and services, as well as for prudential and risk management purposes and, unless you tell us otherwise, to provide you with related marketing information. We also use the information we hold to help detect and prevent illegal activity. We cooperate with police and other enforcement bodies as required or allowed by law.

a Explain why the document is sent to clients.

b Provide reasons for including these statements within the document.

Also included in the document is the following statement:

> We disclose relevant personal information to external organisations that help us provide services. These organisations are bound by confidentiality arrangements. They may include overseas organisations.

c How should the organisation ensure that a client's privacy is protected if they need to send personal details overseas?

SCHOOL-ASSESSED TASK TRACKER

☑ Project plan	☑ Justification	☑ Analysis	☑ Folio of alternative design ideas	☐ Usability tests	☐ Evaluation and assessment	☐ Final submission

9780170440943

APPLY YOUR KNOWLEDGE

3 The Children's Singing and Dancing Academy is an organisation that runs singing, dancing and acting classes specifically aimed at school-age children. The company offers classes after school on most weeknights and on weekends in various locations around Melbourne. Children from many suburbs participate in this extracurricular activity. The company is a not-for-profit organisation set up specifically to broaden children's interest in the performing arts. The Academy has a website and advertises its classes and locations through this medium. Pictures taken of students during their end-of-semester performances are used for advertising the company. The company relies on technology usage by creating brochures, updating their website and storing clients' data. Although the data stored is primarily about the children, data on the parents/guardians of the children is also stored.

- **a** What particular data on children is stored at the Children's Singing and Dancing Academy?
- **b** What particular data on the parents/guardians is stored at the Children's Singing and Dancing Academy?
- **c** Why does the company need to store data on their clients?
- **d** If you were to write an organisational goal for the Children's Singing and Dancing Academy, what would it be?
- **e** What does the Children's Singing and Dancing Academy need to do to ensure they are compliant with the *Privacy Act 1988*?
- **f** What measures are needed for the Children's Singing and Dancing Academy to protect the integrity of data and information?

9780170440943

PREPARING FOR Unit 3

OUTCOME 2

Design, develop and evaluate a software solution that satisfies a client need or opportunity in response to the SRS and present a project plan for managing progress of the project

To achieve this Outcome you will draw on key knowledge and key skills outlined in Area of Study 2. This Outcome concludes the analysis and design stage of the problem-solving methodology for Unit 3, Outcome 2.

Outcome milestones

1. Develop a project plan (Gantt chart). This plan includes all known dates and tasks for Unit 3 and Unit 4. (SAC and SAT dates and tasks will be available from your teacher.) The project plan outlines tasks to be completed, which will include duration, sequencing, milestones and dependencies.
2. Compile an SRS report. The SRS provides an analysis that defines details of the software solution purpose, requirements, constraints and scope, functional and non-functional features, and analytical tools diagrams (data dictionary, context diagram, use case diagram, data flow diagram, flow chart).
3. Design the software solution folio. The design folio includes two or three feasible alternative design ideas, which are rough sketches or mock-ups of development strategies, without much detail. Choose one design idea using criteria derived from the SRS, then add further details to create a complete preferred design of your software solution.

Steps to follow

1. Develop the SRS, including:
   - a. purpose and audience of the SRS
   - b. user characteristics (general characteristics of eventual users)
   - c. environment characteristics (technical description of environment within which the solution will operate)
   - d. scope of the solution
   - e. functional requirements
   - f. non-functional requirements
   - g. constraints
   - h. analytical tools diagrams
     - i. use case diagrams (UCD)
     - ii. context diagrams
     - iii. data flow diagrams (DFD).

SCHOOL-ASSESSED TASK TRACKER

☑ Project plan	☑ Justification	☑ Analysis	☑ Folio of alternative designs ideas	☐ Usability tests	☐ Evaluation and assessment	☐ Final submission

9780170440943

2 Use appropriate software or manual design techniques to design the software solution. Develop two or three feasible alternative designs, and add further details of your preferred design with a total of three or four design strategies. Your design folio should include information about:
   - a WIRE-FRAME of each alternative design, using manual or software methods.
   - b relevant and appropriate evaluation criteria to select the preferred design. These criteria are derived from the SRS.
   - c how the selection of the nominated preferred design from an evaluation of each alternate design was achieved.
   - d the final preferred design, with further developed complete details; may include mock-ups with colours, font, layout specifications for input and output screens, screen sizing, screen layout alternatives (notebook, smartphone, tablet), pseudocode, and data dictionary descriptions.

3 Develop a proposed project plan (Gantt chart), as a written report or annotated visual plan. Note: This will be used as a progress report and will be resubmitted with changes for project evaluation in Unit 4, Outcome 1.
   - a The Gantt chart will include all known dates and tasks, duration, sequencing, milestones and dependencies. A critical path will be indicated. Unit 3 and 4 dates will be provided by your teacher.
   - b Problem-solving methodology stages will be included over both parts of the project. Analysis, design, development and evaluation stages must be explicitly stated and break down tasks included with estimates of duration, sequence and resources..
   - c The project duration and tasks must include both Unit 3, Outcome 2 and Unit 4, Outcome 1.

Documents required for assessment

1 An analysis of the client need or opportunity in the form of a software requirements specification (SRS) that defines the purpose, requirements, constraints, scope, functional and non-functional features and analytical tools diagrams as detailed in Unit 3, Outcome 2

2 A folio of two or three alternative design ideas and the detailed design specifications of the preferred design

3 A proposed project plan (Gantt chart) for monitoring project progress in a written report or an annotated visual plan

Assessment

Your teacher will provide you with a more detailed set of assessment criteria before you begin this assessment.

The SAT (comprising Unit 3, Outcome 2 and Unit 4, Outcome 1) will contribute 30 per cent to your Software Development study score.

9780170440943

Unit 4

INTRODUCTION

In Unit 3, Outcome 2, you identified a client who requested a software solution to their problem or opportunity. The software requirements specification (SRS) provided details about data relationships and functional and non-functional requirements. A design folio outlined a preferred design, chosen from several alternatives, with mock-up, data dictionary and pseudocode descriptions in preparation for further development of a software solution.

In Unit 4, Area of Study 1, you will use the development and evaluation stages of the problem-solving methodology to complete the second part of the School-assessed task (the SAT). This will develop the preferred design, identified in part one, into a software solution using an appropriate programming language.

You will especially focus on good design (chosen from several alternative design ideas) and communicating your message clearly to the intended audience.

You will also evaluate how effectively your software solution achieves its goals.

Throughout the development of your solution, you will use and evaluate the project management plan you began in Unit 3, Outcome 2.

In Area of study 2, you will focus on the security risks to software and data during the software development process. You will do this by examining the security practices of an organisation and the risks to software and data during the development and use of the software solutions.

You will also evaluate the current security practices and develop a risk management plan.

Area of Study 1 – Software Development: Development and evaluation

OUTCOME 1 In this Outcome, you will develop and evaluate a software solution that meets requirements, evaluate the effectiveness of the development model and assess the effectiveness of the project plan.

Area of Study 2 – Cybersecurity: software security

OUTCOME 2 In this Outcome, you will investigate the current software development security strategies of an organisation, identify the risks and the consequences of ineffective strategies and recommend a risk management plan to improve current security practices.

Contains extracts reproduced from the VCE Applied Computing Study Design (2020–2023) © VCAA; used with permission.

Shutterstock.com/whiteMocca

9780170440943

CHAPTER

5 Software development and project evaluation

KEY KNOWLEDGE

On completion of this chapter, you will be able to demonstrate knowledge of:

Digital systems

- procedures and techniques for handling and managing files and data, including archiving, backing up, disposing of files and data and security.

Data and information

- ways in which storage medium, transmission technology and organisation of files affect access to data
- uses of data structures to organise and manipulate data.

Approaches to problem solving

- processing features of a programming language, including classes, control structures, instructions and methods
- characteristics of efficient and effective solutions
- techniques for checking that coded solutions meet design specifications, including construction of test data
- validation techniques, including existence checking, range checking and type checking
- techniques for testing the usability of solutions and forms of documenting test results
- techniques for recording the progress of projects, including adjustments to tasks and timeframes, annotations and logs
- factors that influence the effectiveness of development models
- strategies for evaluating the efficiency and effectiveness of software solutions and assessing project plans.

Reproduced from the VCE Applied Computing Study Design (2020–2023) © VCAA; used with permission.

FOR THE STUDENT

This chapter will cover the theory and skills required for Unit 4, Outcome 1. You will consider procedures and techniques for file management, including how access to data is affected by storage medium. We will discuss control structures as a feature of your chosen programming language as well as characteristics of effective and efficient solutions leading to the construction of test data, validation techniques and usability testing. Finally, the chapter will consider evaluation of the efficiency and effectiveness of the software solution, the problem-solving methodology (PSM) and the project management plan.

By the end of this chapter, you will be ready to develop and evaluate your software solution, and to report on the effectiveness of the PSM and the Gantt chart in monitoring the progress of your software solution.

FOR THE TEACHER

This chapter provides the theory and skills needed for Unit 4, Outcome 1. Having covered analysis and design requirements of the PSM in chapters 3 and 4, students are now introduced to the development and evaluation stages. Students will study file handling and management of files and data, how storage media affects access to data, and data structures for organising and managing data. Programming languages, processing features, characteristics of efficient and effective solutions, techniques for checking that SRS design specifications are verified will be reviewed. Testing, validation and usability techniques will be applied to the software solution. The chapter concludes by considering project evaluation of the software solution, efficiency and effectiveness of the PSM and the project management plan.

By the end of this chapter, students should be able to complete development of their software solution using features of the chosen programming language to effectively manage file storage and handling; document the progress of the project using an updated Gantt chart with annotations; evaluate the efficiency and effectiveness of the software solution with testing, validation and usability testing; and evaluate the effectiveness of the problem-solving methodology and the project management plan.

Note: Students will develop their own software product for the SAT, including identifying the software solution.

Shutterstock.com/William Potter

9780170440943

Managing files

Managing electronic documents forms part of an overall file management strategy. A **comprehensive management plan** will include all aspects of handling documents, including storage, retrieval, backups, archiving and security.

Computer operating systems and applications have wonderful search functions, but these tools are not as efficient as knowing where files are kept. An easily understood filing system allows you to go directly to the folder or file. Good file management practices always save time, whether used by a single person on one computer or in organisations with a few or hundreds of employees.

A file management plan

Establishing a file management plan involves following these steps:

1. Creating the document management plan
2. Implementing the plan
3. Following through and establishing consistency

Creating documents

A business organisation will have many types of documents, such as sales estimates, email, spreadsheets, invoices and publicity brochures. Is there a process for creating these documents? For example, are there standard templates with logo letterhead for regular business documents? If so, where are they located? Is there a style guide to be followed? How are new documents date and time stamped? How will documents be shared?

TABLE 5.1 Comparison of operating system filename limitations

Windows OS	macOS	Linux
Windows operating system (Win32) has a limit, called MAX_PATH, of 260 characters for the file path name; e.g. C:\Program Files\filename.txt. A filename and location may be acceptable until moved or copied to another location where the new path name exceeds 260 characters. Certain characters are reserved by the operating system and are not permitted for general use: < (less than) > (greater than) : (colon) " (double quote) / (forward slash) \ (backslash) | (vertical bar or pipe) ? (question mark) * (asterisk)	There is no filename length limit, though after 1024 characters Finder has display issues. If, however, the file is ever to be shared beyond the host computer, then a 255-character limit ensures compatibility with Win32 computers. There are only two characters that are disallowed by OSX: \ (backslash) : (colon) Note: Office does not allow : (colon) in filenames. However, for cross platform compatibility, the Win32 limitations have become the default standard.	Just one character is reserved / (forward slash). There is no limit on filename length. However, for cross platform compatibility, the Win32 limitations have become the default standard.
These conditions may change; refer to the links for the current restrictions.		
Current restrictions	Cross platform compatability	Linux restrictions

SCHOOL-ASSESSED TASK TRACKER

☑ Project plan	☑ Justification	☑ Analysis	☑ Folio of alternative designs ideas	☑ Usability tests	☐ Evaluation and assessment	☐ Final submission

9780170440943

Naming documents

When naming documents, choose meaningful filenames that can be clearly understood. Dates can also be included to ensure files are displayed in a sorted order. There are some limitations on filenames.

The filename SofDevProjectV1.php has more meaning associated with the name than SDV1.php

This approach also means subsequent versions can easily be identified. Version control is important when projects generate updates of a file. For example:

SofDevProjectV1.php
SofDevProjectV3.php
SofDevProjectV4.php

In a similar way, including the date can indicate when the file was created, without having to open the file or inspect Properties. For example:

ElectricityNMIData20181201
ElectricityNMIData20191201
ElectricityNMIData20201201

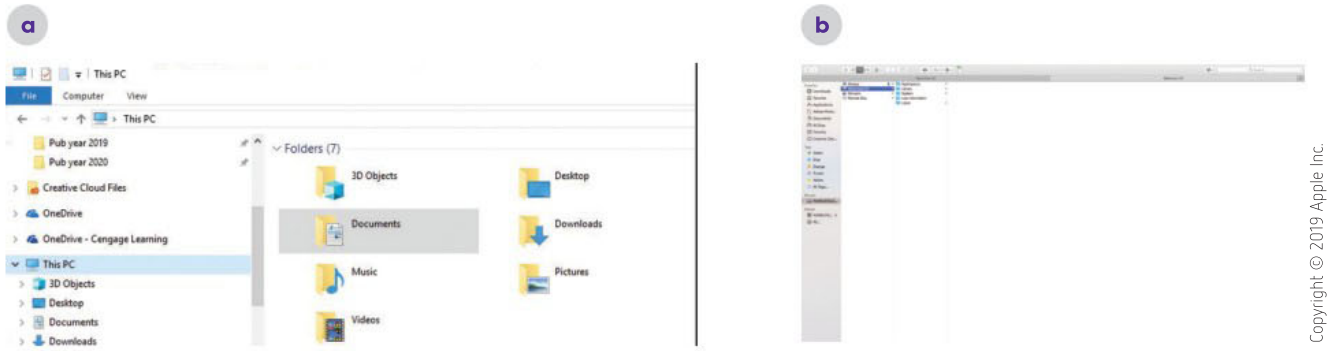
Downloaded on December 1st each year.

Storing and retrieving files within a directory structure

Directory structure is the same as folder structure.

Managing files on a computer requires a consistent method of allocating locations for those files. Directories are the names given to the hierarchy of folders that can be constructed.

Every new installation of an operating system establishes a standard arrangement. Once a user begins creating folders and allocating files, the certainty of where a file may be found is gone.



Used with permission from Microsoft

Copyright © 2019 Apple Inc. All rights reserved.

FIGURE 5.1 a Windows 10 directory; b macOSX directory

Some basic file management approaches that apply to every computer file system are:

- Most user created files should go into the My Documents folder (WIN10) or Documents (macOSX).
- Brevity promotes clarity.
- Use folder names to specify where files are to be saved.
- Keep folder names short, but meaningful.
- Separate work in progress from finished tasks.
- In your regular or monthly clean-up, consider moving files that you no longer need to another drive; you could call that collection 'Archives'.
- Avoid deep and wide folder structures. If there are so many sub-folders that the display is cut off, consider alternatives.
- Keep similar file types together, such as applications, video, music, pictures, graphics, .docx, .xlsx, .pptx etc., as this makes searching simpler.

- Limit the number of files kept. Many files are unnecessary to keep once they have been read and action has been taken.
- Your email inbox is NOT a filing system. Save important messages and attachments to known locations. Choose to delete, move or keep email as soon as you have read the message.

Where there are too many copies of a file, **version control** can be a problem. While one may have been updated, the others will be out of date, despite retaining the same name.

Consider using shortcuts and aliases instead of creating multiple copies that may later become 'orphans' and never be updated. The single source of truth (SSOT) is a management approach that attempts to reduce duplication of files with a similar purpose but different details. For example, a downloaded or printed directory is no longer updated and cannot be changed. Yet that file can be made widely available, putting out inaccurate, misleading or wrong information.

To create a shortcut:	Windows right-click/ make shortcut	macOS ctrl-click /make alias

Once created, just drag and drop the shortcut to another location.

Cloud storage

Cloud storage is now an option to overcome multiple copies of source files and work in progress. A single online data warehouse storage can hold all data files in one place. The backup strategy is the responsibility of the warehouse provider. Examples of free online storage suitable for student use are: Google drive (15GB), Onedrive (5GB), Dropbox (2GB), Box (10GB), iCloud (5GB).

Backups

A 'good' backup must be up to date so data must be 'mirrored'. There will be a tension between how many times a file will be saved, and copied, and the time it takers to save/copy the file. Small file sizes will not be an issue. Larger files will take some time to save and to transmit to the off-site secure location. An alternative strategy is to save only the changed parts of the file. These incremental saves are much quicker than a full save. Specialist software is often used to ensure better efficiency, i.e. less time taken is more efficient. When it comes to backups, follow the 3-2-1 rule. Keep three copies in two formats, with one off-site. If you have just one copy of a file, it can't be that important!

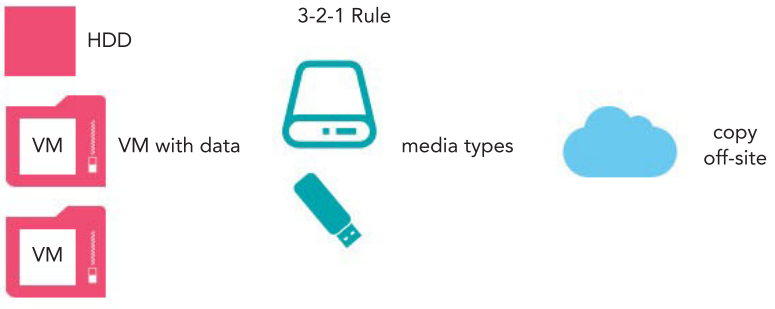


FIGURE 5.2 The 3-2-1 backup strategy

The purpose of a backup is to ensure a copy of the primary data is available in case the original is damaged or lost. Data recovery will take time, and the process and time taken forms part of a **disaster recovery plan (DRP)**.

The logic behind this strategy is that keeping a copy of important data on another partition, such as on the D:\ drive, is not enough, as the media may fail or the computer may

9780170440943

be damaged, stolen or lost, especially if you are using a notebook computer. Another reason to have more than two copies of data is to avoid a situation where the primary copy and the only backup are both stored in the same location. Cloud storage is now considered to be a different medium, and one that is off-site.

An old adage states that, 'There are only two types of hard drive: those that have failed, and those that have yet to fail'.

RAID file storage is becoming more affordable. RAID – Redundant Array of Independent Drives – relies on the probability that 'only' one drive will fail. If all the drives are matched from the same manufactured batch, then a construction defect may cause the drives to be short-lived. **MTBF** – Mean Time Between Failures – is a statistical expectation, and a dramatically shorter or longer time is possible. Reliability of drives is also measured using Annualised Failure Rate (AFR) or Component Design Life (CDL). For consumer devices, a CDL would be five years and an AFR < 0.8 per cent. Alternative media include SD cards, portable HDD, portable SSD, **CDs** and **DVDs**. Floppy drives are no longer considered a viable medium, as access to a read/write drive may be unreliable.

Bathtub curve

All engineered devices have an effective serviceable lifetime during which the performance is fit for purpose. Once the performance degrades, the device must be replaced. Reliability engineering attempts to predict the working lifetime of a device by calculating the failure rate (symbol λ, called lambda, Greek letter 'l'). The MTBF is calculated as 1/λ.

The bathtub curve has three distinct sections:

1. Decreasing failure rate, or early failures, usually due to manufacturing defects
2. Constant failure rate caused by random unpredictable events
3. Increasing failure rate due to the device wearing out

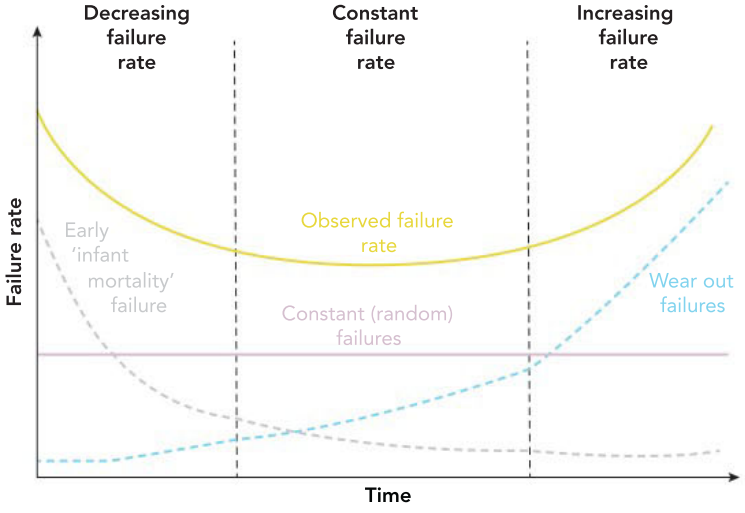


FIGURE 5.3 The 'bathtub curve' estimates the expected failure rate of an engineered device.

A tactic to reduce the first section of early failures is to 'burn in' the device. This requires the device to be run under test conditions until the start of the second section. There are limits to this approach, as the boundaries between sections are not well defined, and can often only be described with any precision after all devices have failed. This strategy is not useful for prediction of future events.

THINK ABOUT SOFTWARE DEVELOPMENT

1. If a HDD is rated with a MTBF of 1.2 million hours, how long is this in days, weeks, months and years? If 1000 drives were operating for eight hours a day, how many would be expected to fail in the first year?
2. Research the drives in your computers. What is the expected MTBF?
3. What is the typical MTBF for a 256 GB micro-SDXC card X10?

Disposing of files

Placing a file in the recycle or trash marks the file to be deleted. The deletion takes place once the recycle or trash is emptied. The space on the drive is flagged as being available. The next time a file is saved, this space may be overwritten.

Deleting a file does not remove the data from the drive or memory. When a file is 'deleted', the data usually remains untouched and the file allocation table (FAT) or table of contents (TOC) entry is removed. This will allow a 'file recovery' application to reassemble the data from the volume so the file can be read. The only change is that the name of the file may have been lost. Many file recovery applications regularly record the Directory structure so that a complete re-build can be achieved.

To permanently delete a file, the data must be made unreadable, also known as being **scrubbed** or **wiped**. This will require the data to be overwritten with 0s and 1s. There are different levels of erasing data. A single overwrite of 1s and 0s is insufficient to remove the magnetic pattern of the original data. File recovery applications can recover data that has been overwritten. Multiple overwrites are necessary to prevent recovery. Three overwrites are considered to be sufficient scrubbing. This requires a cycle of writing all locations to read 1, then overwriting with 0, then repeating this twice more, for three cycles.

Windows: SDelete is a Microsoft command level utility that will securely clean the space on a logical disk. The number of passes can be specified.
macOS: Disk utility application that is integrated within macOS X has a function to erase 1, 7 or 35 times.

Disposing of a drive

When a computer is being passed on to be used by someone else, the existing data should be removed or deleted. The simplest strategy is to replace the drives with new, unused drives. If the drives are to be erased, then a secure erase process of overwriting at least three times must be completed. If the removed drives contain sensitive data, then the drives should be physically disabled or destroyed.

Archiving

It is important to understand that backups are not archives. Remember, the purpose of a backup is to allow a recovery after damage or loss. Archives, however, preserve a record of events and are often required by regulatory authorities. Archives are placed in medium- to long-term storage and are usually compressed to preserve storage space. An index of files archived would help locate the necessary document. For example, the Australian Tax Office requires all records to be kept for seven years, although prosecutions can go back 10 or more years.

The archived data may not be easily accessible to a regular user who is logged in daily. While backups may occur hourly or daily, archiving usually occurs when files are no longer needed, such as each year following the tax reporting season, after a major project is completed and signed off, when a semester unit is finished, or at the end of year.

Factors affecting access of data

File access can be affected by the method of file organisation and the storage media onto which the files are placed. There are different technologies available for storage, either mechanical or solid state.

File organisation and storage media

Frequently accessed files need to be placed in an easily accessible location on fast and reliable media. The choice of media for storage is often about capacity, but other important attributes are:

- latency, or speed of access
- reliability

9780170440943

You can use an Internet search to locate typical latency times, access speeds and costs for:
- HDD
- SSD
- fusion drive.

- cost
- ease of use.

The fastest access can be provided by **SSD**. When placed on SSD, the operating system offers a typical start-up time of 10 seconds, compared to several minutes for **HDD**. Modern operating systems require at least 10 per cent 'scratch space' for saving frequently accessed system files. A hybrid solution is a 'fusion drive', which has a small SSD attached to a HDD.

Organising and manipulating data using data structures

Data structures were first introduced in chapter 1, page 5, for Unit 3, Outcome 1. We will briefly revisit some of the main ideas here. A data structure is a way of organising and storing data for efficient access and operations. There are a number of data structures. Relevant data structures for VCE Software Development include:

- array
- associative arrays, including hash tables and dictionaries
- fields
- records
- queues
- stacks
- lists
- linked lists
- classes
- objects.

Data structure types

Arrays

Arrays have elements that can be any data type. Each element is indexed to uniquely identify it with an integer. The simplest array is a 1-D (one dimensional) array. This stores one piece of data under each index. Multi-dimensional arrays are also possible. Two indices are required for a 2-D array, 3 indices for 3-D, etc. N-dimensional arrays, while possible, may not have any physical equivalent. For example, while a two-dimensional grid can be thought of as having rows and columns, there is also a convention to use i, j, k as indices. The grid co-ordinates would correspond to GRID[i,j]. A three-dimensional matrix could have an array MATRIX[i,j,k].

Arrays are considered when storing and accessing a sequence of objects. For example, if there were six car registrations to be sorted into alphabetical order, the data would first be placed into an array with a data type: string (see Figure 5.4).

Index	0	1	2	3	4	5
Registration	1AB234	1AC712	1BB111	1AB235	1BA101	1AA222

FIGURE 5.4 The array element index often begins at zero, i.e. Registration[0] is the first element.

9780170440943

A sort **algorithm** can arrange the registrations into a sorted order. A search algorithm could locate a specific **value**.

Arrays are well suited to organising large numbers of elements, such as when you want to add 100 numbers together. Without an array, you would input one data value at a time, assign a value to each variable, then add them together and output the result (see Figure 5.5). An alternative approach uses an array Number[n] where n can be from 0 to 99 for a total of 100 elements (see Figure 5.6).

```
BEGIN
      Total ← 0
      Input number1
      Input number2
      Input number3
      .
      .
      Input number99
      Input number100
      Total ← number1 + number2 + number3 + … + number99 + number100
      DISPLAY Total
END
```

FIGURE 5.5 Pseudocode for a simple addition of 100 elements

```
BEGIN
      Total ← 0
      FOR Loop ← 0 to 99
            Input Number(Loop)
            Total ← Total + Number(Loop)
      NEXT Loop
      DISPLAY Total
END
```

FIGURE 5.6 Pseudocode for adding an array with 100 elements

Compare the two pseudocode versions. The non-array version (Figure 5.5) would require 100 lines of code to take the input values. When using an array (Figure 5.6), just six lines of code are needed.

If the task were extended to 500 numbers, the array would still require just six lines of code. The FOR Loop would change from 0 to 99 to become 0 to 499.

In general, an array of size n will have elements from index 0 to n − 1.

Associative arrays

An **associative array**, also called dictionary or map, holds data as pairs. The pairs are indexed using a **key** and a value. The key is unique and may be used no more than once.

Hash tables

A **hash table** is often used to implement an associative array. A hash function generates a key index, which allows the value to be found. The hash function may create the same key index for more than one value. This collision must be handled within the software program. (Refer to chapter 1, page 7 for a detailed explanation and example.)

9780170440943

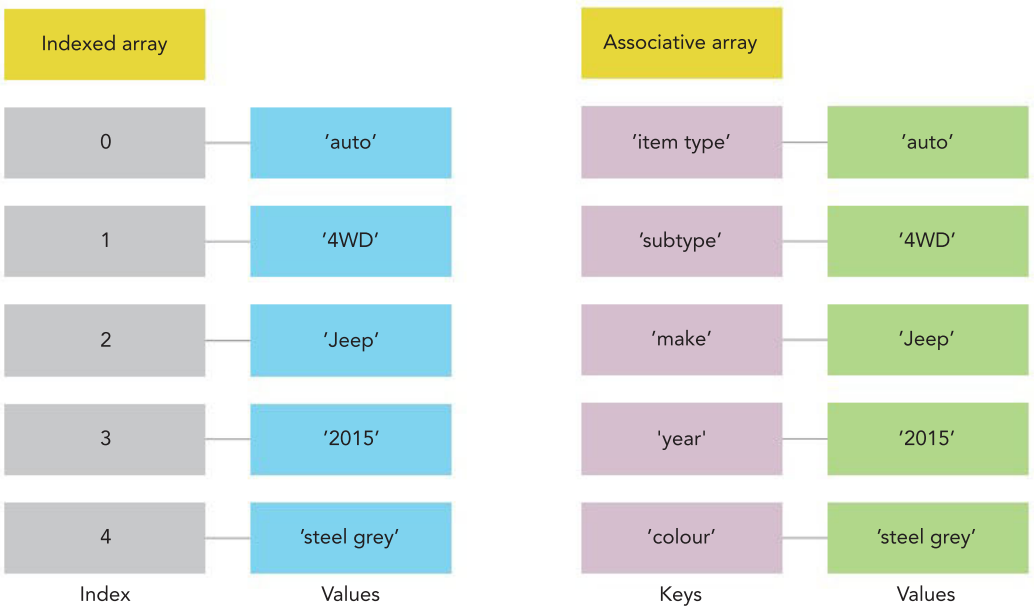


FIGURE 5.7 Comparing indexed arrays and associative arrays

Records

A **record** is a basic data structure for collections of related elements. These elements may or may not be of the same data type. Records are most frequently used in database systems and spreadsheets, but they are also commonly used in programming languages, where they are referred to as **structs**.

A record consists of a number of **fields** that are typically fixed – that is, the fields do not tend to change once the record is defined and used. Each field has a name and each has its own data type. For example, a customer record may contain fields such as firstName, lastName and dateOfBirth.

Records are most useful when a collection of variables are related to each other. The structure provides a logical method of ordering data within a program so that data can be accessed quickly.

In object-oriented programming languages, a record is essentially an object that has no object-oriented features, containing only collections of fields and values. Records and fields can also exist in some types of structured plain text files.

As records contain programmer-defined fields, there are no set operations that can be listed for the record data structure. Rather, there are common operations that can be performed on the record and the fields within it, such as assignment and comparison, as well as adding or removing fields.

Searching and sorting

Once data has been stored in a record or array, the next operation is to search or sort the data.

Search

Searching for data can be accomplished using either linear search or binary search. Linear search is the simpler of the two processes.

Searching and sorting were also covered in chapter 2, pages 46–56.

9780170440943

Linear search

A linear search begins at the first element and compares the value with the search item. If there is a match then a flag 'found' is returned, and the next stage of the program takes place.

If no match is found, then the search moves on to the next element and continues the process until all elements have been compared. The maximum number of comparisons will be n, where n is the number of elements. The average number of comparisons will be n/2.

Consequently, linear search is suitable for collections of a few elements, or a single search on unordered elements.

```
ALGORITHM searchArray()
//Purpose: linear searches through a list of elements
//there are two comparisons per iteration
       //number of elements left to search/finished search
       //search term found
BEGIN
       Input arrayNames
       Input searchTerm

       indexLenNames ← length of arrayNames
       FOR i ← 0, i < indexLenNames, i ← i + 1 DO
              IF arrayNames[index] = searchTerm THEN
              PRINT "Found" + searchTerm
              ENDIF
       ENDFOR
END
```

FIGURE 5.8 Pseudocode for linear search

Binary search requires a sorted collection

Binary search is more efficient than linear search. As a divide and decrease algorithm, binary compares terms very quickly. As the number of elements increases, the advantages are dramatic. Binary search has been discussed in more detail, with examples, in chapter 2 (see Figure 2.27, page 54).

For small collections, the combined time of sorting then binary searching may take longer than the linear search. If the collection will be searched multiple times, however, then the binary search becomes the better choice.

Sort

There are a large number of sorting algorithms available. For our purposes, the simpler ones are quicker to understand and implement. For beginning software developers, quicksort, selection sort and bubble sort provide a solid basis to explore other, more complex, algorithms.

Bubble sort

This simple sorting algorithm is not efficient for a large number of elements. For example, a one-dimensional array of numbers has five elements. To sort the array using bubble sort from lowest number to highest will require three passes (see Figure 5.10).

9780170440943

```
ALGORITHM binarySearch(arrayNames, searchTerm)
// Purpose: binary searches through a list of elements
// Inputs: an array of elements to be searched
//         and the item being searched for
// Output: Boolean, True if item found, False if not found
BEGIN
        found ← FALSE
        indexLenNames ← the length of arrayNames
        midPt ← the middle index value of arrayNames
        IF searchItem = arrayNames[midPt] THEN
                found ← TRUE
        ELSEIF indexLenNames > 1 THEN
                IF searchItem < arrayNames[midPt] THEN
                    low ← first index value of arrayNames
                    RETURN binarySearch(arra yNames[low to midPt-1],
                                                searchItem )
        ELSEIF searchItem > arrayNames[midPt] THEN
                  high ← indexLenNames
                  RETURN binarySearch(arra yNames[midPt to high],
                                              searchItem)
                  ENDIF
        ENDIF
        RETURN found //false
END
```

FIGURE 5.9 Pseudocode for a binary search

```
First pass:
[7 4 5 3 9] → [4 7 5 3 9] first comparison, 7 > 4, swap positions, now 4
7
[4 7 5 3 9] → [4 5 7 3 9] second comparison, 7 > 5, swap positions, now
5 7
[4 5 7 3 9] → [4 5 3 7 9] third comparison, 7 > 3, swap positions, now 3
7
[4 5 3 7 9] → [4 5 3 7 9] fourth comparison, 7 < 9, no change
Second pass
[4 5 3 7 9] → [4 5 3 7 9] first comparison 4 < 5, no change
[4 5 3 7 9] → [4 3 5 7 9] second comparison 5 > 3, swap position now 3 5
[4 3 5 7 9] → [4 3 5 7 9] third comparison 5 < 7 no change
[4 3 5 7 9] → [4 3 5 7 9] fourth comparison 7 < 9 no change
Third pass
[4 3 5 7 9] → [3 4 5 7 9] compare 4 > 3, swap, now 3 4
[3 4 5 7 9] → [3 4 5 7 9] compare 4 < 5 no change
[3 4 5 7 9] → [3 4 5 7 9]
[3 4 5 7 9] → [3 4 5 7 9]
```

FIGURE 5.10 Example of bubble sorting algorithm process. During the third pass, the array was sorted after the first comparison, but the algorithm did not 'know' this until the entire pass had been completed.

The process is a simple comparison: if greater, swap; if not, move to the next element. This process is repeated, passing through the population until there are no further swaps.

The only positive factor in favour of bubble sort is its simplicity; it is very easy for beginning programmers to understand. By any other measure, bubble sort cannot be recommended.

9780170440943

```
ALGORITHM bubbleSort(Array)
//Purpose: comparison sort through a list of elements
BEGIN
       n = indexLenArray
       REPEAT
             swapped ← FALSE
             FOR i = 0, n-1 DO
// if this pair is out of order
             IF A[i] > A[i+1] THEN
// swap them and remember something changed
             swap( A[i], A[i+1] )
             swapped ← TRUE
             ENDIF
          ENDFOR
       UNTIL swapped ← TRUE
END
```

FIGURE 5.11 Bubble sort is not efficient or effective, but it is simple to follow the steps.

Quicksort

Quicksort is a **comparison sort** algorithm that is also known as a divide and conquer algorithm (see chapter 2, page 50). A pivot value is chosen, and all values that are smaller are moved before it, and all values that are larger are moved after it. These sub-lists can then be sorted into a single sorted list.

```
ALGORITHM quicksort(Array, low, high)
//Purpose: partition sort through a list of elements
//Array has elements from A[0] to A[n]:length of array= n + 1
//initial values of low and high
     //high = indexLenArray; length of array - 1
     //low = 0
//Call this function using
//quicksort(arrayNames, 0, indexLenNames - 1)
BEGIN
  IF low < high THEN
       p ← partition(Array, low, high)
       quicksort(Array, low, p - 1)
       quicksort(Array, p + 1, high)
END
//function to determine partition
// i is index of lower partition
ALGORITHM partition(A, lo, hi)
  pivot ← A[hi]
  i ← lo
  FOR j ← lo, hi - 1, i ← i + 1 DO
       IF A[j] < pivot THEN
       swap A[i] with A[j]
       ENDIF
  swap A[i] with A[hi]
       ENDFOR
RETURN i
```

FIGURE 5.12 Example of pseudocode description for Quicksort

The slowest performance is caused by placing the pivot as the first or last element in the list. A well-placed pivot can increase the speed of the sort. In practice, a randomly chosen pivot provides acceptable performance compared with other slower algorithms, such as bubble sort.

9780170440943

Quicksort is not considered a **stable sort**, as the relative order is not maintained for equal sort items after sorting.

Selection sort

There are many ways to carry out a sort; so far we have discussed two simple mechanisms. A third simple and fast algorithm is **selection sort** (see chapter 2, page 47). The name describes the process of repeatedly selecting the next smallest element and swapping places. The index of the last smallest value increases each time a swap takes place. In this way, the entire population is compared until all elements are in order of size.

Figure 5.13 shows an example of selection sort, where six unordered elements are considered.

Sorted sublist	Unsorted sublist	Smallest element in the unsorted sublist
( )	(32, 12, 27, 65, 26, 33)	12
(12)	(32, 27, 65, 26, 33)	26
(12, 26)	(32, 27, 65, 33)	27
(12, 26, 27)	(32, 65, 33)	32
(12, 26, 27, 32)	(65, 33)	33
(12, 26, 27, 32, 33)	(65)	65
(12, 26, 27, 32, 33, 65)	( )	-

FIGURE 5.13 Selection sort begins at the left and progresses through the list

A similar process can be applied to a list or a **linked list**. Linked lists were introduced in Chapter 1. This process can remove the smallest element and insert at the end of the sorted values.

65 32 12 27 26 33	Begin with element 0, left-most element, find the smallest and insert
12 **65** 32 27 26 33	Begin with element 1, repeat, find the next smallest and insert
12 26 **65** 32 27 33	Repeat, search from element 2, insert
12 26 27 **65** 32 33	
12 26 27 32 **65** 33	
12 26 27 32 33 **65**	When the search index equals the length of the list, the process stops.

FIGURE 5.14 Selection sort of an unordered list

Insertion sort

Insertion sort is a simple sorting algorithm that builds the sorted list one element at a time. On short lists and small arrays (less than 25 elements), insertion sort will perform fewer comparisons than selection sort, depending on the particular 'sortedness' of the list or array.

CASE STUDY

9780170440943

FIGURE 5.15 When sorting a hand of cards, the player usually selects one card and compares it with each other card in turn before placing the cards into sorted order. Usually card players arrange the cards smallest to largest, from left to right.

```
ALGORITHM insertion-sort(Array)
//length of array is n; elements A[0 .. n-1]
BEGIN
        n = indexLenArray
        FOR j ← 1 TO n - 1 DO
        key ← Array[j]
        //insert element A[j] into the sorted sequence A[0 .. j]
        i ← j - 1
        WHILE i >= 0 AND A[i] > key DO
              A[i + 1] ← A[i]
              i ← i - 1
        A[i + 1] ← key
END
```

FIGURE 5.16 Pseudocode for insertion sort

5.2 THINK ABOUT SOFTWARE DEVELOPMENT

1. Create an insertion sort algorithm that arranges smallest to largest, right to left.
2. Compare and contrast to identify the similarities and differences between insertion sort left to right and right to left.

The following is an example of how to use insertion sort to place five cards into increasing numerical order.

1. Applying the pseudocode algorithm to a player who has dealt a hand with five cards. For example:

7	2	10	5	4

Scope:

The array length is 5 (n = 5)

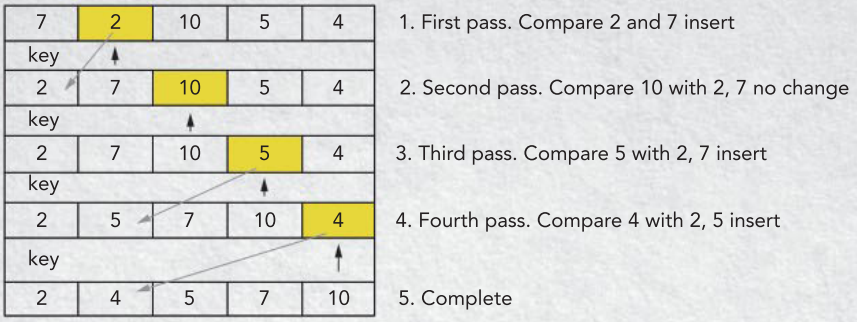
The maximum number of passes through the array to sort all cards will be n-1 or 4 passes.

2. Assign index to array elements:

A[0]	A[1]	A[2]	A[3]	A[4]
7	2	10	5	4

9780170440943

3 Ignore the first element, as it has been assumed to be sorted. Begin with A[1]:



As well as writing the code, you will need to check other important details in order for your program to operate correctly. Testing of your code must be documented in a test table. Validation of data and data types are to be thoroughly carried out.
Remember, testing is not validation. One checks that the code executes as expected. The other checks boundary conditions for data so the limitations are observed.
Be sure to know the difference, as a program can execute correctly with the wrong input data.

Features of a programming language

Programming features were first introduced in chapter 2 (see page 35). This section will review and extend on that knowledge for inclusion in your SAT Unit 4, Outcome 1. Stage 3 of the problem-solving methodology (PSM) is development. Your development task will implement your previously chosen design, with details documented in your client's software requirements specification (SRS).

Choice of programming language

Your actual programming language will be chosen at your school in accordance with Software Development programming requirements. These requirements are considered each year, and any changes are announced annually in the VCAA Bulletin.

The discussion about languages presented in this text provides development within three conceptual layers:

- *Interface* includes graphical user interface (GUI) for digital systems such as notebooks, tablets, smartphones, gaming consoles, and microprocessors including robots.
- *Logic* is applied through data structures, data validation techniques, and the control structures of selection, iteration and sequencing. Modular organisation and code optimisation is supported. The programming language contains objects, methods and event-driven programming functions.
- *Data sources* can allow retrieval of data from external storage or external access technologies.

There are many programming languages used within education, in industry and for personal use. No two languages are the same. Programming languages come with their own syntax and style, but there are elements in a computer language that can be found in all languages.

The following are features that every programming language contains.

Instructions and syntax

Every language has its own syntax. **Syntax** is the set of rules that are used to create instructions. An instruction is something you want the computer to execute. A language reference guide is usually available to assist in finding the syntax and reserved words for the language.

9780170440943

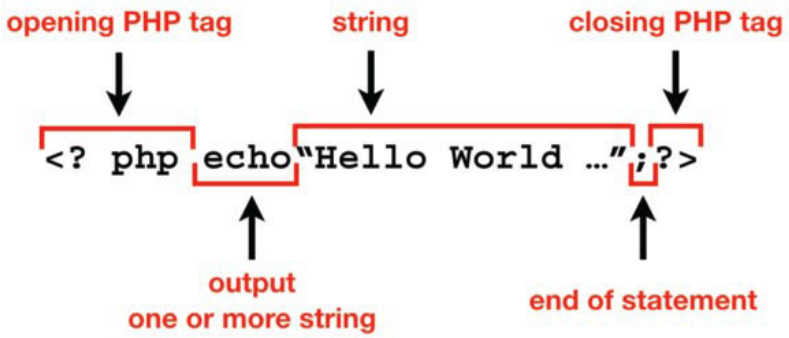


FIGURE 5.17 An example of PHP syntax

The following shows syntax with a couple of instructions:

```
if ($my_name == "someone")
    {
    echo "Your name is someone! <br/>";
    }
echo "Welcome to my homepage!";
```

The words in bold are the built-in commands for this particular language, while the whole lines (and there are three lines) are instructions and follow a specific syntax.

TABLE 5.2 Examples of Hello World! in several computer languages

JAVA	`public class Main`
	`{`
	`  public static void main(String[] args)`
	`  {`
	`  System.out.println("Goodbye, World!");`
	`  }`
	`}`

PYTHON	`Print ("Hello, World!")`

C#	`public class Hello`
	`{`
	`  public static void Main()`
	`  {`
	`  Console.WriteLine("Hello, World!");`
	`  }`
	`}`

VB – visual basic	`Console.WriteLine("Hello, World!")`

HTML (note: HTML is NOT object oriented and does NOT meet VCAA requirements)	`<!DOCTYPE html>`
	`<html>`
	`  <head>`
	`  <title>Hello, World!</title>`
	`  </head>`
	`  <body>`
	`  <p>Hello, World!</p>`
	`  </body>`
	`</html>`

PHP	`<?php`
	`echo "Hello, World!";`
	`?>`

RUBY	`puts 'Hello, World!'`

9780170440943

TABLE 5.2 Examples of Hello World! in several computer languages (continued)

Mathematica	`CloudDeploy[ ExportForm[ Style[ Framed["Hello, World", ImageMargins -> 60], 80, Orange, FontFamily -> "Verdana"], "GIF"], Permissions -> "Public"]` Once written, Wolfram language uses CloudObject, which does not require a 'host computer', just a browser.

CloudObject

There are many other programming languages, but any language chosen in VCE Software Development must meet the **VCAA programming requirements**.

Control structures

Control structures were introduced in chapter 2 (page 35) and several detailed examples were provided. Here is a brief review.

Control flow is the execution order of statements, instructions or functions that are evaluated in a program. Flow of control is implemented with three basic types of control structures:

- sequential
- selection, which chooses between two or more alternatives
- repetition or iteration, where looping determines how many times the instruction is executed.

Sequence or sequential structure

Sequential is the default mode, where instructions are executed line by line in the sequence they were written.

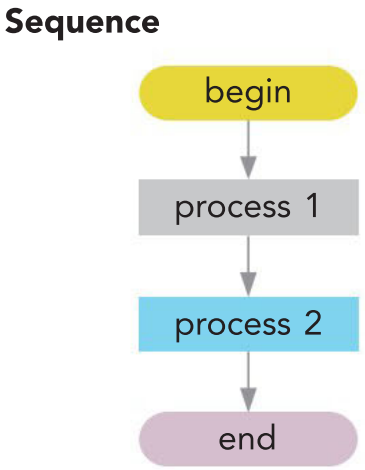


FIGURE 5.18 Start at the beginning and follow the steps, one by one.

9780170440943

Selection structure

Selection is used for branching when a choice is made between two or more alternate paths. All the instructions for every program can be constructed from just seven control structure statements (see Table 5.3).

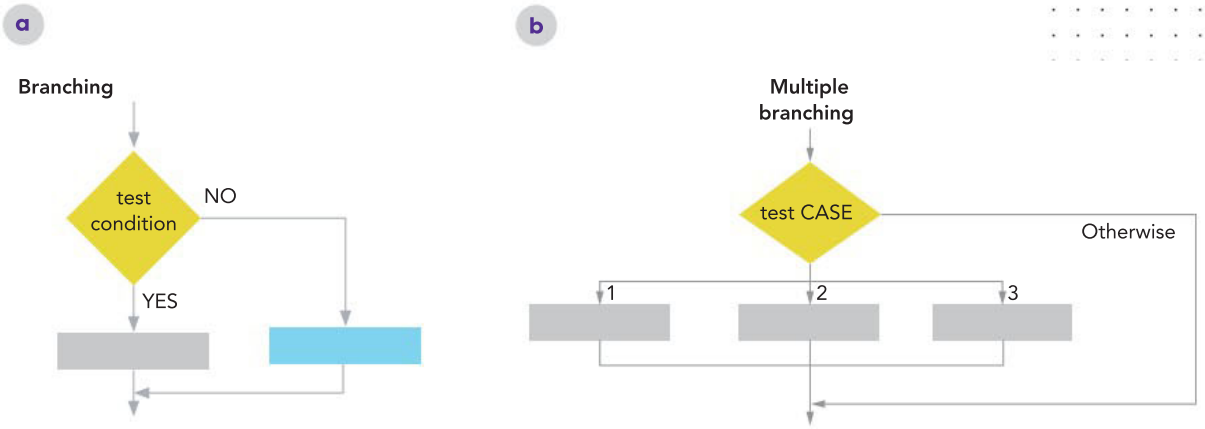


FIGURE 5.19 Branching selection: **a** Binary, e.g. If/Then; **b** Multiple branches, e.g. Switch or Select. CASE.

Iteration structure

Repetition allows loops to execute a section of code multiple times. There are three types of loops.

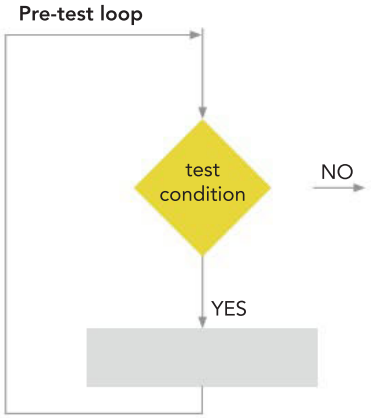


FIGURE 5.20 Pre-test: the condition is met at the beginning of the loop, e.g. While-Do

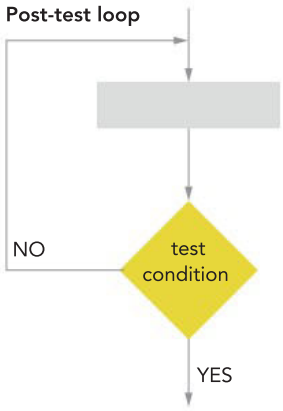


FIGURE 5.21 Post-test: the condition is met at the end of the loop, e.g. Repeat Until

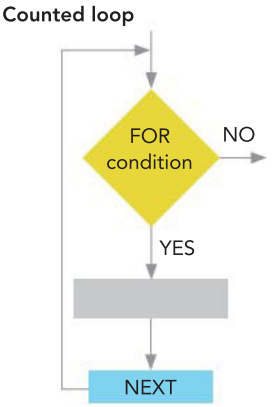


FIGURE 5.22 Counted where the number of times can be specified

9780170440943

TABLE 5.3 Control structure statements

Control structure statement	Action
If	When condition is met, flow is directed to the alternative instruction
Switch or Case	Choose between listed alternatives
If ... Then ... Else	When condition is met, flow can be directed multiple times
While ...	May not execute to satisfy condition
Do ... While	May not execute if condition is not satisfied once
Repeat ... Until	Executes at least once If conditions are not met then there is a possibility of an **infinite loop**
For	A counting loop to be executed a set number of times

These control statements are combined in only two ways: **stacking** and **nesting**. The stacking method for control statements has the exit point of one control statement connecting to the entry point of the next control statement. These control statements are simply lined up one after the other, or stacked. The only other method for connecting control statements is to place control statements inside other control statements. This method is known as nesting.

Methods, functions and procedure subroutine

It is important to understand the difference between a function, a subroutine and a method. To identify each one, just remember:

- if a subroutine returns a value, then it is a function
- if a subroutine does not return a value, then it is a procedure
- if the function is coded within a class, then it is a method.

Functions*

*Depending on the programming language you have chosen to use, these may have different names. For example, in PHP they are all functions.

A function is a block of statements that takes one or more inputs and then executes those statements and returns a value. A function is mainly used when a programmer requires a block of instructions to be used frequently in the program.

There are two types of functions:

- built-in functions
- user-defined functions.

Built-in library functions

Built-in library functions are functions that are created as part of the language. These built-in functions are there to make programming easier. For instance, most languages have a date function, which would return the computer's current date.

Make sure you check the handbook or online programming advice websites for information about what you specifically want a function to do before you create your own function. There is usually a long list of optimised built-in functions that have been fully tested.

9780170440943

Used with permission from Microsoft

Function	Description
DATE function	Returns the serial number of a particular date
DATEDIF function	Calculates the number of days, months, or years between two dates. This function is useful in formulas where you need to calculate an age.
DATEVALUE function	Converts a date in the form of text to a serial number
DAY function	Converts a serial number to a day of the month

FIGURE 5.23 An example of Excel built-in functions.

© 1990 – 2014, Python Software Foundation

Function	Description
abs()	Returns the absolute value of a number
all()	Returns True if all items in an iterable object are true
any()	Returns True if any item in an iterable object is true
ascii()	Returns a readable version of an object. Replaces none-ascii characters with escape character
bin()	Returns the binary version of a number
bool()	Returns the boolean value of the specified object
bytearray()	Returns an array of bytes
bytes()	Returns a bytes object

FIGURE 5.24 Python built-in functions listed at www.W3schools.com

Efficient and effective solutions

Efficiency and effectiveness can often be seen as competing goals. A process may be effective if it is thorough and checks all possible options. A similar task may be considered efficient if it is quick, uses few resources and is completed with minimal interactions with other activities. For a task to be more effective, it will often become less efficient.

Efficiency is gauged in terms of measurable quantities. Each part of your software solution will need data collected relating to these quantities:

- time taken
- the effort required
- resources accessed.

Once this data is collected, you can evaluate how efficient your solution may be.

Effectiveness relates to accuracy. What characteristics would a software solution need in order to be effective? The output has to suit the client's purpose and be in a format that follows appropriate conventions as described in the SRS. The information produced by the system needs to be accurate and comprehensive, without glaring omissions or superfluous information.

9780170440943

Techniques for checking coded solutions

There are some basic and essential software testing processes that every software developer should be performing as they go, before the code is shown to anyone else.

- Functionality testing means every feature is tested at each stage and after any changes.
- Versions are kept of working code, files are saved and backed up regularly with date (and time if necessary). This may be automated in some development environments.
- Undertake one change at a time, and ensure it is tested. When change is working with no errors, the next change/improvement can be introduced.
- Undertake module testing, where independent modules can be modified without affecting other parts of the software solution.
- Identify edge condition or boundary values for testing while coding, and place them into the test table for later.

Different types of testing

Unit tests are low level and test close to the source of your application. Individual methods and functions of the classes and components of your software solution are tested.

Integration tests verify that the different modules used by your application are working together. Multiple parts of the solution must be up and running for this test to be worthwhile.

Functional tests are only interested in whether the SRS are met. The output is checked, and not intermediate stages.

The difference between an integration test and a function test may be that integration just verifies that the database can answer a query, while a function test will expect a specific value.

Acceptance testing is a formal test that verifies whether the SRS has been satisfied and the client will accept the final version of the software.

Performance testing will observe response times, user behaviour and how the system behaves in a production setting. Performance testing often indicates where further improvements can be made to the software.

Other types of tests

Black box testing observes inputs and outputs only. The internal process is unknown.

White box testing considers processes within the component to achieve the correct output.

User acceptance testing determines whether end users are satisfied. A combination of interviews, questionnaires and observations can determine end user sentiment, which may not be visible through one technique.

Validation techniques

Validation checks that input data are reasonable. Validation does not and cannot check that inputs are accurate. For example, it cannot tell whether a person is being honest when entering their age. However, validation can detect problems when a person enters their age as 174 years, or 'banana', or nothing at all. You can perform validation manually (yourself) or allow software to do it for you.

9780170440943

Computers are particularly good at conducting validation checks.

1 *Existence checks* ensure that a value has been entered and the field is not blank, or <null>.
2 *Type checks* ensure data is of the right type; for example, the age that has been entered is actually a number.
3 *Range checks* ensure that data is within acceptable limits (for example, children enrolling in kindergarten must be 3–6 years old) or comes from a list of acceptable values (for example, small, medium or large).
4 *Format checks* ensure that data is in the correct format. For example, a birth date should be numbers in the format 00/00/0000.
5 *Consistency checks* perform a comparison between different entries. For example, users enter their email address twice. Survey responses are often cross-checked to ensure the responses are based on fact, and not made up.
6 *Reasonableness checks* consider whether the details are plausible. For example, a 10-year-old would not have a driver's licence.

People can perform manual validation, such as proofreading for sense, clarity, relevance and appropriateness. In addition, unlike spreadsheets, people tend to smell a rat when values pass electronic validation checks but are clearly inaccurate because they are ridiculous.

Similarly, Microsoft Word can detect words that are not in its dictionary, but it cannot advise writers that a paragraph is boring or that the text on the previous page was pretentious, misleading or irrelevant.

FIGURE 5.25 Validation rules in FileMaker database. Here an ID field is made compulsory ('Not Empty') and unique, and within a defined range of values. The database is also told what error message to display if validation fails.

Testing

Usability of solutions

Thorough and careful **testing** is necessary, whether the software solution is a game, a website shopping cart, or an airliner autopilot. If a software solution fails, this could disadvantage users.

If your solution fails because of undiscovered faults, it may become difficult to use, or completely unreadable.

Testing checks that a solution will produce the correct output and will do what it should do. Testing is not simple, quick or cheap – especially for a product such as an operating system with megabytes of code in thousands of files created by hundreds of people.

The typical steps involved in testing are as follows:

1 Decide which tests will be conducted.
2 Create suitable test data.
3 Determine expected results.
4 Conduct the test.

Verification or validation?

Verification checks that the specific requirements are being developed. Validation checks that the final product meets expectations.

THINK ABOUT SOFTWARE DEVELOPMENT 5.3

If a user is expected to enter their age, what would be a reasonable range check?

a 5–50 years
b 15–80 years
c 0–100 years
d 1–200 years

THINK ABOUT SOFTWARE DEVELOPMENT 5.4

In Unit 4, Outcome 1, which data will you need to validate, and how will you achieve this?

Remember:

- **validation** checks whether the data inputs are reasonable
- **testing** checks the accuracy of information outputs.

In your SAT, ensure that all of your data is thoroughly and appropriately validated.

SCHOOL-ASSESSED TASK TRACKER

☑ Project plan	☑ Justification	☑ Analysis	☑ Folio of alternative designs ideas	☑ Usability tests	☐ Evaluation and assessment	☐ Final submission

9780170440943

5 Record the actual results.

6 Correct any errors.

7 Document all steps in a summary table.

There are many test types, each intended to uncover different kinds of errors at different times during development. The types of testing relevant to your solution are listed in Table 5.4.

TABLE 5.4 Testing types

Name of test	What is tested?
Informal (alpha)	The part of the solution that has just been finished
User acceptance (beta)	Typical end users use their own equipment to check that the finished solution is acceptable under different user conditions
Component	A single part of a system works properly by itself (for example, a user entry form applies the correct delivery cost for a given destination postcode)
Integration	Individual parts of a system work together (for example, the embedded code correctly accesses the separate database table)
System	All components in the solution work properly as a single unit
Installation	The form control is installed correctly and is working on your system, server or domain
Compatibility	The code and its components are compatible with a variety of computers and the main OS
Usability	Whether users can operate your software solution quickly and simply
Accessibility	Whether users with special needs or disabilities can use your software solution

Test data

To prove the accuracy of the output of a software solution, provide some test data to work with, and compare the actual output answer with expected output that is known to be correct.

Good test data includes:

- valid data – data that is perfectly acceptable, reasonable and fit to be processed.
- valid but unusual data – data that should not be rejected even though it seems odd. For example, a 10-year-old might, very occasionally, enrol in university. Validation that rejected the young genius' enrolment would cause embarrassment.
- invalid data – to test the code's validation routines. For example, if people must be 18 years old to be given a credit card, test data should include people under 18 so they can be seen to be rejected.
- boundary condition data – data that is on the borderline of some critical value where the behaviour of the code should change. These 'tipping point' errors are a frequent cause of logical errors in programming.
- wrong data – data with an inappropriate format would be expected to generate an error.
- absent data – a blank field entry will test how the system handles a 'no entry' entry.

TABLE 5.5 A data test table with suitable (a) dummy data (b) boundary test

	Data name	Data type	Range	Below range	Within range	Above range
(a)	Postcode	Numeric	3000–3999	2200	3500	4500
	ID number	Numeric	500–2000	200	1000	3000
	Order number	Numeric	00100–10000	50	500	11000

9780170440943

	Data name	Data type	Normal data range	Below lower boundary	At boundary	Above upper boundary
(b)	Postcode	Numeric	3000–3999	2999	3000 and 3999	4000
	ID number	Numeric	500–2000	499	500 and 2000	2001
	Order number	Numeric	00100–10000	99	100 and 10000	10001

Testing your solution

After designing and building your software solution you need to demonstrate that it has been thoroughly tested. You need to know what to test in your solution. These will be discussed in the following sections.

Inputs

Every input must be inspected to check that it is displaying in the right place, at the right time, and at the right speed and volume in a variety of common environments (meaning different browsers and devices).

Buttons and links

Every internal and external button or link in the solution needs to be manually clicked and the result noted. Create a list of buttons and links and tick off each one as it passes testing.

Links to external services

You should be able to completely test all parts of the solution under your control. You need to test the operation of any external connections to your product, to ensure that data updates the function as expected.

Readability

Use the checklist provided in Table 5.6 to test readability of your solution.

TABLE 5.6 Readability checklist

Checklist	
Is the text large enough to read comfortably on a small device?	
Is contrast optimal, or at least satisfactory?	
Is the typeface a readable size?	
Are all buttons labelled and identifiable as buttons?	
Is text alignment attractive and readable on the page?	
Are the spelling, punctuation and grammar correct?	
Is expression clear and unambiguous?	

Calculations

If your solution calculates any information, its answers need to be verified by manual recalculation in a testing table. For example, you might create a screen containing code to display a countdown timer to the next Software Development examination. To prove that you have tested the accuracy of its output, take a screenshot. Annotate the screenshot with whatever

9780170440943

manual calculations will demonstrate that it is true, based on the time the screenshot was taken and the time of the exam.

Loading times

If the software solution is online, clear any cache to remove pre-loaded copies of files and media and try loading the site via cable and wi-fi. Any page that takes more than a few seconds to load should be inspected and optimised. Another method is to use one of many online services that can measure the loading times for your pages. Online data repositories may have varying access times due to user demand.

Accessibility

Making the web accessible

Does your solution create unnecessary difficulty for users with common disabilities such as poor eyesight or muscular control, or other challenges such as weak language skills? Is alt text applied to images? Are colour combinations considerate of colourblind people? Many colourblind-safe palettes are documented online.

There are several places online to test the accessibility of your solution. Try the Making the web accessible website.

Dynamic features

Every selection option item must be checked and its behaviour documented in a testing table (see Table 5.7). If data entry forms are expected to work, data should be entered and its successful arrival at its destination should be documented. Any simulated functionality, such as a faked login box, should, as far as is practical, appear to work genuinely. Any coding should be run using a variety of test data and the behaviour of the code recorded.

Usability testing

A usability test is undertaken with real users, to determine how easy the software solution is to use. A researcher observes users while they complete tasks with the software. Any opinion about the software is recorded. Whether problems are met and dealt with, and any frustration, confusion or ambiguity is also noted. If several users are observed, any similarities can be identified as programming or user issues. All will need to be simplified and removed in order to improve the usability. Ideally, these changes will be accomplished before the software is released.

Elements of a test plan

Identify each of the following and state your specific details.

Scope

- State what will be tested

Purpose

- Test questions, concerns and goals

System requirements

- Hardware and software necessary for the test

Users

- Identify who the interview participants will be

Details

- State the questions that will be asked
- Identify the quantitative data that will be collected
- Summarise any likes, dislikes and suggestions for improvement

FIGURE 5.26 Planning a usability test

9780170440943

Conducting a usability test

A formal interview with the user will achieve consistency if more than one person is to be interviewed and observed.

1. Establish how long the testing might take.
2. Emphasise that the interview is about the software product and not the user's ability to operate the software and the equipment.
3. Consider a video recording. This recording could be consulted to provide further data, but must not be used for any other purpose. A signed consent form will be required if video is to be recorded.
4. Explain the use of equipment and software.
5. Introduce each task, for the video record, and observe the user's responses. Ask the user to 'think aloud' to ensure their actions are clear.
6. At the completion of the tasks, thank the user for their time and invite any comments, questions or observations about the usability test.

Documenting test results

A **testing table** is a commonly used way to record evidence of functionality testing. A testing table for a software solution may look like the example in Table 5.7.

TABLE 5.7 A testing table

What was tested	How it was tested	Expected result	Actual result	How it was fixed, if relevant
Age < 18	17 y 364 d	'under age'	'under age'	-
Age = 18	18 y 0 d	'happy birthday'	'under age'	Boundary condition test was fixed
Age > 18	18 y 1 d	'adult'	'adult'	-
Readability	Asked two volunteers to read sample pages and report on text size, contrast, alignment, spelling, vocabulary, offensiveness and headings	Reports each page was easy to read, accurate and inoffensive	One reader suggested a button label in italics was hard to read	Changed button label text style from italics to bold
Code calculation of number of days since the last event	Set computer's clock to 7 days after the event	Index page should display 'The last event occurred 7 days ago.'	Displayed '... created 6.89586 days ago ...'	Rounded up the age calculation in the code

How to document your testing

- Use a testing table such as the one shown in Table 5.7.
- Seek a subjective report from a fellow student who tried out your solution's readability and usability.

9780170440943

- Capture screenshots of features that are not normally visible, such as dropdown menus and warning messages, showing that they work when needed.
- Make handwritten calculations, annotating printed screenshots of your solution's calculations to verify that the output has been checked for accuracy.
- Capture screenshots of the solution's validation rules, responding properly to invalid data.

Recording the progress of projects

Adjustments to tasks and time frames

In the first half of this SAT (Unit 3, Outcome 2), we have discussed how to manage a project, and why project management is important. In Unit 4, you will continue to use these project management techniques to complete development and evaluation of your software solution.

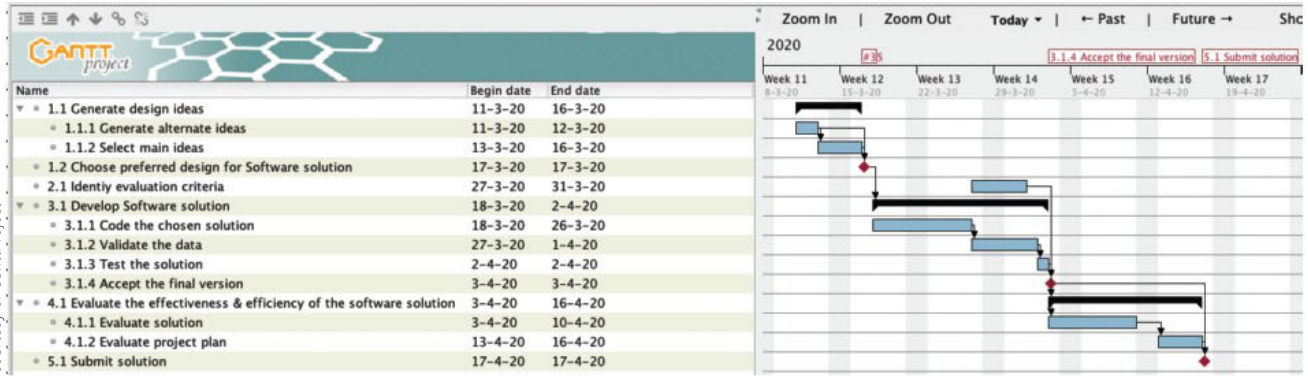
Very few projects ever proceed perfectly in line with the project plan. One unexpected rainy day, a hard disk crash or a sick day for a key worker can slow down a work team enough to affect tasks, other teams and deadlines. Project plans are not written in stone, and Gantt charts should be regularly modified to reflect reality.

Project plans are living documents. Tasks that run overtime may have resources added to them or be modified so they finish earlier. Bad weather may force changes to scheduling, so indoor tasks may be completed instead of outdoor work. Late deliveries of equipment may cause a project manager to move people off one task and assign them to another one that can proceed without the deliveries.

Annotations

A project plan should be annotated to give reasons for changes to task schedules or resourcing priorities. When the project is later assessed, these annotations will serve as valuable lessons before undertaking the next project. Annotations might also be added by other project leaders to advise the team of significant news or concerns. Annotations could be handwritten or added as notes in the Gantt chart itself.

NelsonNet additional resource: Figure 5.27



Courtesy of GanttProject

FIGURE 5.27 Gantt chart: a project plan for the chosen solution

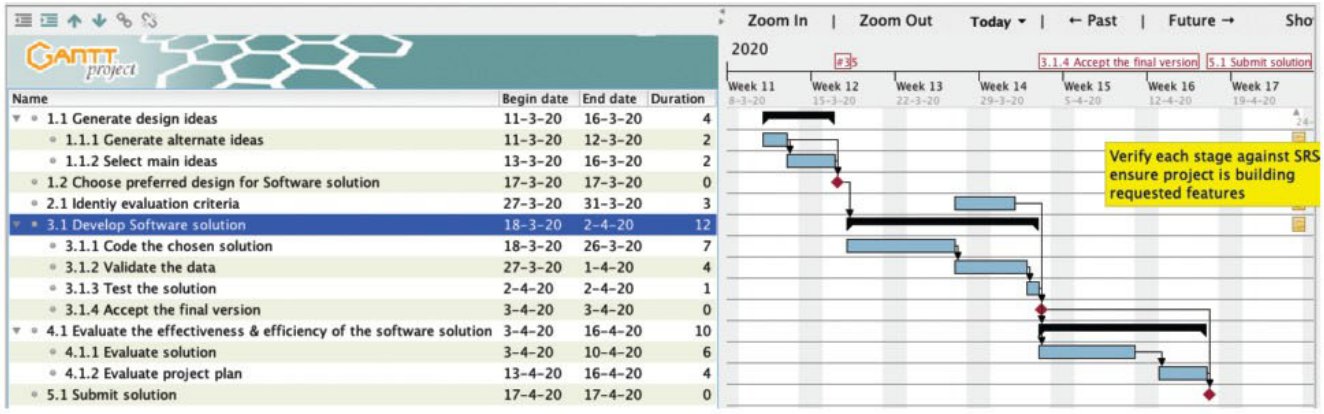
9780170440943

RESEARCH

Gantt chart

1. Modify the project plan by adding two days to the '2.1 Identify evaluation criteria' stage.
   Redraw Figure 5.27, either by hand or electronically, or modify the electronic file on student NelsonNet.
2. Do you need to adjust the rest of the project so that the final date for Submit solution does not move forward by more than one day?
3. Justify how you have maintained the project end date.

Handwritten annotations are also acceptable. One strategy for Unit 4, Outcome 1 is to print out the final version of the Gantt chart and write directly onto the printed copy. Laminating before annotation adds extra presentation finesse. Printing to PDF and annotating that is also an option, or you could just place a screenshot into Powerpoint and add callouts. Annotation needs to be quick, simple and effective. Where do the comments and observations, insights and suggestions come from?



Name	Begin date	End date	Duration
1.1 Generate design ideas	11-3-20	16-3-20	4
1.1.1 Generate alternate ideas	11-3-20	12-3-20	2
1.1.2 Select main ideas	13-3-20	16-3-20	2
1.2 Choose preferred design for Software solution	17-3-20	17-3-20	0
2.1 Identiy evaluation criteria	27-3-20	31-3-20	3
3.1 Develop Software solution	18-3-20	2-4-20	12
3.1.1 Code the chosen solution	18-3-20	26-3-20	7
3.1.2 Validate the data	27-3-20	1-4-20	4
3.1.3 Test the solution	2-4-20	2-4-20	1
3.1.4 Accept the final version	3-4-20	3-4-20	0
4.1 Evaluate the effectiveness & efficiency of the software solution	3-4-20	16-4-20	10
4.1.1 Evaluate solution	3-4-20	10-4-20	6
4.1.2 Evaluate project plan	13-4-20	16-4-20	4
5.1 Submit solution	17-4-20	17-4-20	0

Courtesy of GanttProject

FIGURE 5.28 Gantt chart with notes added

Keeping logs

Project logs are a record of all the small and large steps a project takes on its way to completion. The log is usually in electronic form and is shared with all project leaders. It may be created using specialist log software, in a **weblog** ('blog') for example, or with Microsoft Word or Excel. It could be shared online using Google Drive or similar technology.

Conditional formatting

The spreadsheet has many inbuilt functions that automate the formatting of the appearance. Once initial settings have been chosen and tested, the conditionally formatted cells can be copied down the column, ready for the data to be placed.

9780170440943

Automated Excel data validation

Select the cell for the dropdown menu.
Go to Data tab on the ribbon, then Data validation.
On the Setting tab, in the Allow box, click List.
Click in the Source box, then select your list range, or a comma separated list will provide option choices for that cell.

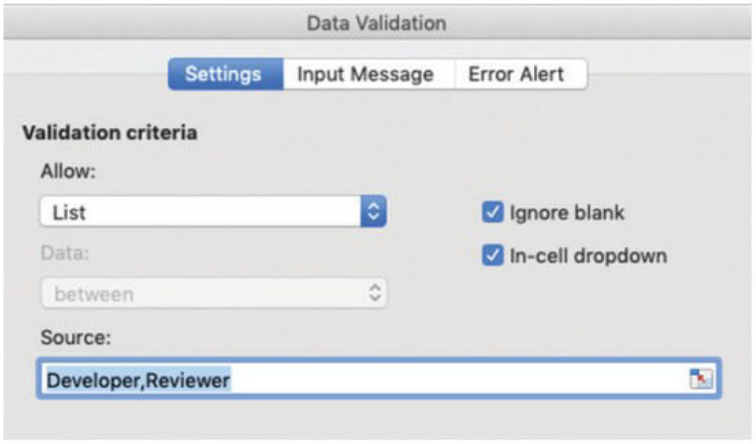
	A	B	C	D	E	F	G	H	I	J
1	Issue #	Date	Reported by	Item/Functionality	Description	Issue type	Priority	Impact	Suggested fix	Status
2	1	2020-05-09	Developer	Input box submit button	Spacing too close to text box	Usability	Medium	High	Increase spacing introduc colour and label with meaning	Resolved, adjusted size, colour and labelling
3	2	2020-05-09	Developer	Title	The font size of the main input screen is too small, hard to read.	Design	High	High	Adjust the size and colour contrast	Resolved, larger font, darker colour
4	3	2020-05-09	Developer	What to do is not clear	After reading the introduction screen, the search and sort buttons are not clearly marked.	Usability	Medium	Medium	Adjust the size and colour shading of the button. Add a label	Partly resolved, need to work out how to include a label inside
5	4	2020-05-09	Developer	Image quality of LOGO	Image is very poor quality	Data	Medium	Low	Locate better original and re-capture better quality screenshot. Completely redraw to create new original.	Still looking to locate image, yet to be attempted.
6	5	2020-05-10	Reviewer	Sources and Privacy statement	There is no acknowledgement of sources, destination of data or privacy statement	Data	Low	Low	Add Sources citation link and link to Privacy terms and conditions to lower left of opening screen	Yet to be completed

Used with permission from Microsoft

Additional resources

NelsonNet additional resource: Figure 5.29 Project log template

FIGURE 5.29 Project log template created in Microsoft Excel. Columns C, F, G and H are data validation dropdown lists. Columns G, H and J also have conditional formatting applied to indicate priority and impact by colour: low, medium, high, critical and showstopper. This template is available for download through your NelsonNet login.



Used with permission from Microsoft

FIGURE 5.30 Excel data validation dialog

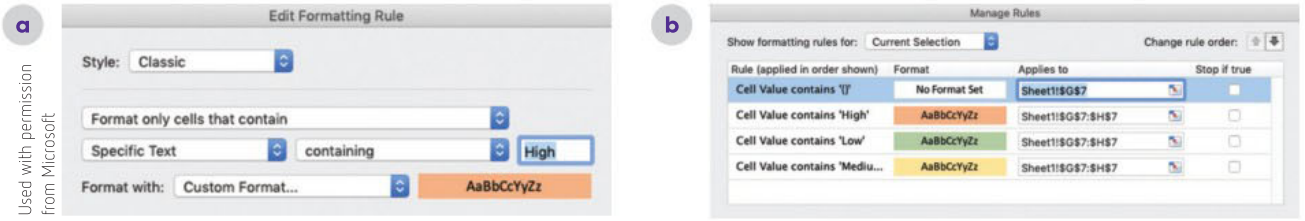
As shown in Figure 5.31, the priority cells can be conditionally formatted to give you a dropdown menu choice of options. High, Medium and Low data will appear once you begin typing, but the formatting of the cells relies on you manually formatting each individual cell. Alternatively, format the first, then copy format and pasting across all similar cells. In this example, the first row is conditionally formatted for Column C: Reported by [Developer, Reviewer], then column F: Issue type [Usability, Design, Data,], column G: Priority [High, Medium, Low], column H: Impact [High, Medium, Low], column J: Status [Resolved, Partly resolved, Not resolved].

A project log helps a team keep track of the status of project tasks, which team member is responsible for the tasks, and when deadlines and milestones are due. It can also help you keep track of the work you are doing on your solution alone. In conjunction with your Gantt chart, a project log can be a valuable tool to help you manage your project efficiently.

9780170440943

It can include time and date stamps, comments on progress, project risks, issues that arise, ideas for solutions, actual solutions, explanations of decisions, results of testing, forecasts and warnings, task changes, photos and future action needed. A project log is like a diary that records the complete history of a project. An online version would be to keep a weblog, or blog. Keeping a weekly blog that records daily achievements would be a very effective method of gathering evidence. Your final Gantt chart can be annotated with entries from your blog, with references to the URL.

If you find yourself struggling to divide up the tasks by priority, creating a priorities quadrant may also be of use (see Figure 5.32).



Used with permission from Microsoft

FIGURE 5.31 An example of automated cell formatting in Excel. **a** New rule; **b** Managing rules

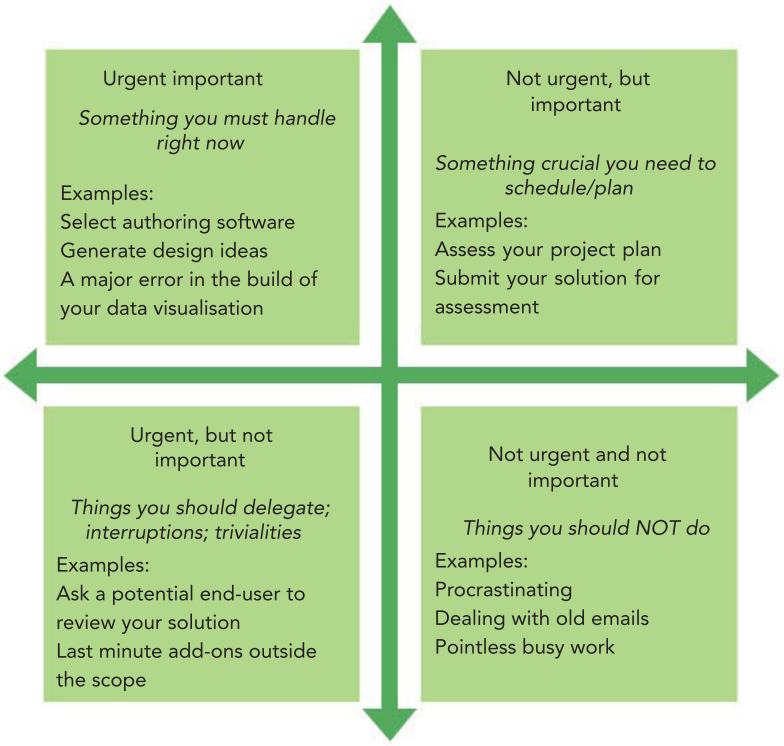


FIGURE 5.32 Example of a priorities quadrant

Efficiency tips

You have a limited amount of time to complete the project, so you need to ensure that you use your time as efficiently as possible. Your first priority is to make your proposed solution work. While the attractiveness of your solution is an assessment criterion, appearance is worth much less than the quality of its information. You need to make sure that you allocate your time appropriately to reflect this priority.

9780170440943

One way of doing this is to use a plain text heading and then insert all of the key information. If you have time later, you can go back and design a graphic heading and modify the fonts. Remember, a beautiful solution with incomplete content is a waste of time.

It is easy to get sidetracked 'fiddling' with small details. Spending half an hour in Photoshop cutting a figure from its background is an inefficient use of your time, and this will be reflected in your assessment. You need to complete the less exciting jobs, such as typing data and testing that all links work.

It may help to make an agreement with yourself to do just a little of the task. It will not be finished, but at least you will have made some progress. This will boost your self-confidence, and you may even decide to do a little bit more while you are at it. The hardest part of most jobs is taking the first step. Once the first step has been taken, it is usually much easier to continue.

Factors influencing the effectiveness of the development model

As previously described, effectiveness is about accuracy. Factors that influence the effectiveness of your software solution will affect the ability of the software to deliver the expected output.

Hardware limitations can limit a program's ability to perform as expected. This may be processing speed limitations, RAM capacity or storage access limitations. Large amounts of data take a finite time to be transferred, processed and stored. These limitations must be factored in when performance expectations are being considered.

Evaluating the efficiency and effectiveness of solutions and project plans

Evaluation

Evaluation is not the same as testing; its purpose is very different.

Evaluation is the final stage of the problem-solving methodology. It checks how well the solution is satisfying the needs of the user it was originally created for.

Remember, you will be evaluating your programming solution and your project plan as part of your assessment in Unit 4, Outcome 1. When evaluating your solution, you need to refer to the evaluation criteria you developed during the design phase. For each criterion, you will choose a method to evaluate it.

Evaluation is not the same as testing; its purpose is distinctly different. By the time evaluation begins, the solution has already been proved to work properly and its functionality is no longer in question.

Evaluation can best be understood by saying what it does not do:

- Evaluation does not test that a solution is working properly. That should have been done during testing.
- Evaluation does not enter test data to check that output is accurate. That should have been done during testing.

9780170440943

- Evaluation does not use a stopwatch to time how long a process takes. That should have been done during testing.
- Evaluation does not perform checks with immediate results, such as pulling out the power plug to see if a system loses data. That should have been done during testing.

Evaluation looks at a solution's performance over time in terms of the evaluation criteria.

What to evaluate

Evaluation criteria are determined during the design phase of the problem-solving methodology, and are based on the most important qualities that the solution is expected to have when it is designed. For example, for your solution, essential criteria include the ability to quickly enter and record data. You should evaluate the features that would, if they were not achieved, render your solution unsatisfactory.

Evaluation criteria fall under two headings: efficiency and effectiveness.

1. Efficiency can be measured in terms of speed or productivity (work produced in a given time), profitability (income generated versus running costs) and labour requirements (how much labour is required to achieve its productivity levels).
2. Effectiveness includes completeness, readability, attractiveness, clarity, accuracy, accessibility, timeliness, communication of message, relevance and usability (see chapter 4, Table 4.2).

You may conduct interviews or create questionnaires to seek feedback from your peers regarding the criteria you were expecting to achieve. This feedback could be offered as evidence of evaluation.

For your software solution, you may decide to use annotated screenshots to demonstrate how key criteria were satisfied, such as readability, or consistency of formatting. If recordings of animated screen activity are needed to prove that you *satisfied* an evaluation criterion, you could use screen recording software such as Camtasia.

Evaluation methods

For each evaluation criterion for your solution, there must be a corresponding evaluation method that can measure the degree to which the criterion has been achieved.

- Objective (fact-based, measurable) results are solid facts that are hard to argue with. Measure whenever you can.
- Subjective results (emotions, opinions, personal judgements) can be gained from interviews, questionnaires and surveys. These should only be used when objective measurement is not possible or practical, such as when evaluating how comfortable users feel when using an unfamiliar programming solution.

Remember: Evaluation assesses your solution's performance over time. It is not instantaneous, like testing is. Any emotional or judgemental feedback is only gathered on appropriate criteria. For example, it is pointless to ask interviewees questions such as, 'Is the new system faster than the old one?' Even if you received an answer to this question, you would not be able to trust its accuracy.

When to evaluate

Evaluation occurs after the solution has been in regular use for some time, when it is well 'bedded in' and its users are familiar and comfortable with it. A few months of regular, daily use is typical.

9780170440943

TABLE 5.8 Typical evaluation methods

Criterion	Method
Accuracy (effectiveness)	Check the complaints log and count the complaints from staff or customers about inaccurate information received from the system over the past three months.
Reliability (effectiveness)	Count the number of faults in the system's error log.
Security (effectiveness)	Count the number of successful and thwarted attempts made to penetrate system security.
Attractiveness, pleasure, comfort, confidence (effectiveness)	Interview users.
Productivity (efficiency)	Refer to system logs and count how many transactions the system handled over three months compared to the previous system.
Profitability (efficiency)	Ask the accountants to tally the new system's running costs over time. Check organisational profit figures and see if profit has increased.
Labour requirements (efficiency)	Count the number of staff hours spent operating and maintaining the system compared with the previous system.
Ease of use, usability (effectiveness)	Count the number of times the help file was used (indicating that the solution may not have been intuitive). Add up how many errors were made by users. (A solution that is hard to use tends to cause users to make mistakes.) Check the help desk records to see how often users asked for help or complained about the solution. Ask users to complete a questionnaire about their feelings regarding the system's usability.

Evaluating a solution too soon can lead to negative feedback, because users may not be used to it, and may be slow and prone to making errors. Later, when they are comfortable and skilled with the solution, their feedback is likely to be more positive.

If a system is used infrequently but its success is critical to the organisation (such as creating school reports, or managing a flood of tax returns at the end of the financial year), evaluation may be done immediately after the system is used.

You will not have the luxury of waiting for a long period of time for users to explore your solution. Instead, you will need to ensure that you evaluate your own solution especially thoroughly, and take into account all of the feedback you received during beta testing, which should have included asking potential end users to look at your solution. (These could have included classmates, friends, your teachers, your parents/guardians and other family members.)

Solutions

Evaluation of your software solution began with the SRS. That document recorded all the client's expectations and specifications. Those performance statements can be converted into evaluation criterion. It is against each of those criteria that the evaluation will be performed.

For example:

- Specification (SRS) statement: The menu will be easy to understand.
- Evaluation criteria: Is the menu easy to understand?

9780170440943

A Likert (pronounced lick-ert) scale would be the most appropriate conversion of opinion to a rating scale. To extend the authority of such a user survey, several users need to be interviewed or questioned about the usability of the software solution.

There may be too many specifications contained within the SRS, so you will need to prioritise which are the most significant.

Project plans

Organisations invest a great deal of time, money and labour into projects, so they tend to look back at their project plans to evaluate how the planning went. Organisations often need to undertake further projects, so they need to evaluate earlier projects to avoid repeating mistakes. Evaluating an organisational project plan can help answer the following questions:

1. Did the project finish on time?
2. What tasks delayed your project? Why weren't these delays anticipated?
3. Could lessons be learned to help the next project finish on time?
4. Did the project finish on budget?
5. What assumptions did we get wrong?
6. Why did this task cost far more than expected? How can we avoid that next time?
7. Why were new requirements being added just weeks before the system was due to go online? Was our analysis a failure?
8. Why did the first three prototypes blow up? Was the design team under-skilled, overworked, under-equipped or working to an impossible deadline?

5.5 THINK ABOUT SOFTWARE DEVELOPMENT

Identify the questions from the list opposite that could also be used to evaluate your project plan.

Even a failed project can be a valuable learning experience if it keeps the same mistakes from happening again. For example, in November 2012, the US Air Force scrapped a major enterprise resource planning software project called the Expeditionary Combat Support System (ECSS) after it cost $1 billion in expenses but failed to create any significant military capability. ECSS was supposed to replace more than 200 **legacy systems**. The project had been underway since 2005, and its ballooning costs clearly suggest that Air Force officials and systems contractors conducted an overwhelming amount of additional custom coding and integration work. An Air Force spokesman said the project would require another $1.1 billion just to complete 25 per cent of the original scope, and that the project would not be complete until 2020.

On the other hand, not undertaking projects to improve systems can also be an expensive mistake. As of 2012, Long Beach, California was owed $18 million in parking fines as a result of antiquated software used by the local government. Fines had gone uncollected because of the age of the systems in use. Staff were stuck using manual processes, which were so time-consuming that there was no time left to undertake collection efforts – leading to $18 million in unpaid fines. Long Beach first knew about the problem in 2009, but still failed to address it.

Your project may be on a more modest scale than these scenarios, but there are still lessons you can learn from them. The purpose of assessing your own project plan is to judge how your plan and the techniques you used to make adjustments along the way helped you manage your project. For instance, how much notice did you ultimately take of your plan? How did your plan assist you when things didn't go as expected? Did the quality of your annotations or other tools you may have used help keep you on track? These types of questions are of great use, both now and for your future studies and later career, because they will help you understand how planning functions alongside real-world projects.

9780170440943

You need to fully think through the scope of the tasks to ensure you don't discover a forgotten task when it is too late to finish it. Verification is important – keep referring to the project plan and SRS so you don't forget tasks, do them in the wrong order, or fail to observe milestones.

You also need to think carefully about your software choices. For example, it would be unwise to choose an editing tool, only to discover halfway through development that it is unsuitable for the task. If you need to learn and practise new software skills, do this before the project begins. Learning new software while you work on a critical solution is very inefficient, and you will be prone to making errors.

You also need to be realistic when you are estimating time requirements. Think back to times when you had similar tasks to complete. Remember which problems caused delays in completing those tasks, and add time to your schedule to deal with them this time. If a task looks like it will run overtime, refer to your Gantt chart to see what consequences this delay will have on the project. If the task has slack built into it already, it may have no effect on subsequent tasks. If it is a critical task, however, you will need to find a way to save time on other tasks.

When evaluating your project plan, you first need to establish the evaluation criteria that will tell you how successful it was in getting the project completed on time.

Such criteria may include the following.

- Completeness: Were any significant tasks omitted from the project management plan? Were resources included? Was it annotated when required?
- Maintainability: How easy was it to modify the Gantt chart to keep it up to date with reality?
- Accuracy: Were tasks correctly identified and marked as dependent or concurrent? Were tasks in the right sequence? Were time estimates realistic?
- Readability: Was it easy to see all tasks and their dependencies? Was the Gantt chart and its text of a readable size? Were colour choices appropriate?

Once your criteria have been chosen, you will again need a method to evaluate each one. Annotated printouts highlighting key features of the software solution may be useful. You might take screenshots before and after changing the duration of a task to show how easy it was to maintain the chart. You can describe how well the project plan worked.

In your project management review report, you need to be able to explain factors that influenced the effectiveness of the project plan. These factors include tasks, due dates, resources, tasks done concurrently, tasks that are dependent on other tasks, resources and people. Your report could also:

- explain the advantages and disadvantages of using a Gantt chart
- offer lessons you learned from this project that will make later projects even more successful
- explain what further development of your software solution might take place for Version 2.0 if you had additional time and resources.

Next steps

Work your way through the chapter summary material, including 'Preparing for Unit 4, Outcome 1', then, after consultation with your teacher's instructions, begin to prepare your solution for submission.

9780170440943

5 CHAPTER SUMMARY

Essential terms

acceptance test a formal test to verify that the SRS for the software solution has been met and that the client will accept the final version

algorithm a series of steps or instructions that achieves a purpose

array a list of elements indexed by position. In most programming languages the first element has index zero.

associative array similar to an array; information is stored in key-value pairs

black box test a test of only inputs and outputs of the software solution

CD compact disk, typically 740MB

comparison sort two values are compared for higher or lower value and placed in order by swapping places if necessary. The comparison sort cycles through the entire list several times.

comprehensive management plan includes storage, retrieval, backups, archiving and security

disaster recovery plan (DRP) details of the steps required to recover the information systems in the event of damage or loss

DVD dynamic versatile disk, typically 4.7GB or 8.5GB with dual layers

evaluation the final stage of the problem-solving methodology. It checks how well the solution is satisfying the needs of the user for which it was originally created.

evaluation criteria performance criteria made from the expectations and specification

fields elements in a record, struct or database

functional test a test to verify that the SRS for the software solution has been met; only output is checked

hash table a data structure that uses a hash function to map keys to values by computing an index that is related to, but smaller than, the initial key

infinite loop when the control structure iteration (loop) has no exit

insertion sort a simple sorting algorithm that builds the sorted list one element at a time

integration test a test to verify that the different modules of the software solution are working together

HDD hard disk drive; a magnetic platter with a read/write mechanism

key used in associative arrays to specify values

legacy system old technology that is still in use

linked list a linear data structure where each element is stored separately and has a pointer to the next element

MTBF mean time between failures; a statistical estimate of the expected reliability of a computer component. Usually read/write actions for drives.

nesting when control statements are placed within other statements. The outer statement waits until the inner statement has finished execution.

performance test a test for response times, user behaviour and system behaviour of the software solution in a production setting

record a fixed number and sequence of related fields, or data, held within one structure. They may be different data types.

scrubbed when a file is permanently deleted; the data must be made unreadable

selection sort a simple sort that repeatedly selects the next smallest element and swaps places

9780170440943

5 CHAPTER SUMMARY

SSD solid state drive; has no moving parts. SSDs have an estimated 'lifetime' that is dependent on the number of read/write actions.

stable sort a sort in which equal sort items have relative order, which is maintained after sorting

stacking when control statements are listed one after another. Each statement is executed in sequence.

structs records used in database systems and programming languages

syntax set of rules used to create meaningful statements in a computer language

testing checks the accuracy of information outputs

testing table a commonly used way to record evidence of functionality testing

unit test a low level test of individual methods and functions of the classes and components of the software solution

user acceptance test a test of whether users of the software solution are satisfied; conducted through interviews, questionnaires and observations

validation checks the reasonableness of data inputs

value the element that is stored as an element in an array

VCAA programming requirements advice considered annually, advising programming language requirements that must be met before using for Software Development

version control the method that keeps track of the current, most up-to-date document through a drafting process

weblog/blog an online diary written in reverse chronological order

white box test a test of processes within the software solution to ensure correct output

wiped when the data is scrubbed and overwritten with 1s and 0s

Important facts

1. A **comprehensive management plan** will include all aspects of handling documents, including storage, retrieval, backups, archiving and security.
2. Clearly understood, meaningful file names are always best.
3. Use shortcuts and aliases to folders and files instead of creating multiple copies, which may later become 'orphans' and never be updated.
4. The purpose of a backup is to ensure a copy of the primary data is available in case the original is damaged or lost. Data recovery will take time, and the process and time taken forms part of a **disaster recovery plan (DRP)**.
5. A full data **wipe** is when a full write and rewrite of 1s and 0s is completed at least three times to ensure the data cannot be recovered and can be considered secure.
6. There are three types of control structures: **sequential**, **selection** and **iteration**.
7. **Archives** are placed in medium- to long-term storage and are usually compressed to preserve storage space. Archives preserve a record of events and are often required by regulatory authorities.
8. A **data structure** is a way of organising and storing data for efficient access and operations.
9. **Arrays** store a fixed number of elements of the same data type.
10. The array element index often begins at zero.
11. An **associative array** is more informative than an (indexed) array. Information is stored in key-value pairs.

9780170440943

12 **Linear search** is suitable for collections of a few elements, or a single search on unordered elements.

13 **Binary search** uses a divide and decrease algorithm, and is more efficient than linear search.

14 There are a number of sorting algorithms available, including **quicksort**, **selection sort** and **bubble sort**.

15 A **sorting** algorithm is **stable**, if the order of equal elements in the input array remains unchanged in the output array.

16 **Syntax** is the set of rules that are used to create instructions. An instruction is something you want the computer to execute.

17 Flow of control is implemented with three basic types of control structures: **sequential**, **selection** and **repetition** or **iteration.**

18 A **routine** returns a value when it is a function; otherwise, it is a procedure.

19 A **function** coded within a class is a **method**.

20 Validation checks that input data are reasonable. There are three types of validation checks: **existence checks, type checks** and **range checks**.

21 Testing of your code should be documented in a testing table.

22 A **project plan** should be annotated to explain reasons for changes to task schedules or resourcing priorities.

23 **Project logs** are a record of all the small and large steps a project takes on its way to completion.

24 To divide up tasks by priority, create a **priorities quadrant**.

25 **Evaluation** checks how well the solution is satisfying the needs of the user for which it was originally created.

26 Evaluation looks at a solution's performance over time in terms of the **evaluation criteria**.

27 Evaluation criteria fall under two headings: **efficiency** and **effectiveness**.

28 For each **evaluation criterion**, there must be a corresponding method that can measure if the criterion has been achieved.

29 **Objective** results (fact-based, measurable) are solid facts that are hard to argue with.

30 **Subjective** results (emotions, opinions, personal judgements) can be gained from interviews, questionnaires and surveys.

31 For **project plan** evaluation, establish criteria that indicate how successful the project plan was. For example: completeness, maintainability, accuracy and readability.

9780170440943

TEST YOUR KNOWLEDGE

Review quiz

Managing files

1 Why do files need to be organised? Explain how a directory hierarchy might work.
2 You are asked for advice on how to keep your software application and data safe and secure. What would you recommend?
3 How can a HDD be securely wiped? How is this different to just deleting a file?

Organising and manipulating data using data structures

4 Explain the difference between an array and a record. Include reference to data type.
5 When is a data dictionary useful for programming? Explain how you would convince someone who thinks the programming language will keep track of variable names and datatypes.

Features of a programming language

6 Briefly explain how any programming language works. Include reference to control structures.
7 What is iteration? Why is it useful? Give an example using an array with ten elements.

Efficient and effective solutions

8 Effectiveness is often subjective. Who will you ask to determine the effectiveness of your software solution? How will you ask so you know they didn't just make up their response?
9 Efficiency is usually measured. What can be measured to demonstrate the efficiency of a software solution?

Techniques for checking coded solutions

10 What are some techniques for checking code?
11 Explain how 'black box' testing occurs. How is that different to 'white box' testing?

Validation techniques

12 How is validation different to testing?
13 Create a table listing what validation does and does not do.
14 List three types of validation.

9780170440943

TEST YOUR KNOWLEDGE

Testing

15 If you cannot 'test everything', what do you test? Explain why.

16 What is a 'testing table'? What does it record?

Recording the progress of projects

17 Why are developers strongly advised to keep an accurate daily log journal?

Factors influencing the effectiveness of the development model

18 The PSM has been referred to many times, in which each of the stages has specific tasks to be completed. Effectiveness might be seen as the ability of the developer to deliver the promised software. Why is a structured development model helpful in achieving the development plan?

19 List some of the advantages and disadvantages of the 'single pass' PSM. Can you suggest an alternative?

Evaluating the efficiency and effectiveness of solutions and project plans

20 What are some of the reasons that a plan may not finish on time?

21 How does a critical path assist with modifying a project plan? Could you have arranged your plan to make better use of the available time?

22 What advice would you give to someone who is thinking of starting their Unit 3 Gantt chart?

9780170440943

APPLY YOUR KNOWLEDGE

1 How would you apply validation to the following user input?
   a Country of birth
   b Age
   c Agreement to terms and conditions
   d Email address
   e Victorian residence address (give two methods)
   f Date of birth (give two methods)
2 Write a pseudocode algorithm for a new member to enter a username and a new password. Include verification of the password, without displaying the characters. Provide a 'Re-enter password' message if the passwords do not match, and an acknowledgement message if there is a match.
3 How would you adjust the sort algorithms to sort from largest to smallest?
4 Write the following as pseudocode, then select and apply suitable test data for the following data entry:
   a A person is an adult if their age is 18 years or older.
   b Data entry of daily maximum and minimum temperatures in Victoria
   c Ages of clients for a local dental surgery
   d Car registration number plates
   e A system accepts values between 1 and 20 entered as input.

9780170440943

PREPARING FOR Unit 4

OUTCOME 1

Develop the software solution, using an approved programming language, conduct user evaluation in response to the SRS criteria. Evaluate the project plan for managing progress.

On completion of this unit, the student should be able to develop and evaluate a software solution that meets requirements, evaluate the effectiveness of the development model and assess the effectiveness of the project plan in monitoring progress.

This outcome (U4O1) will be assessed according to the school-based assessment advice issued annually by the VCAA. All details should be verified each year prior to beginning your submission. Your teacher will provide specific details of the assessment rubric that will be used in that year.

The methods for the usability testing, evaluation of the efficiency and effectiveness of the software solution and the project plan evaluation have been developed throughout chapter 5. Long reports must follow a sequence and have internal consistency. The examples and suggestions in chapter 5 provide some guidance on how to accomplish the PSM – development and evaluation stages to complete Unit 4, Outcome 1.

There are four criteria assessed in Unit 4, Outcome 1. Teachers will provide you with a schedule for submission of these assessments.

The following completed assessment tasks are to be submitted for the SAT:

- Client software solution
- Usability testing
- Evaluation of efficiency and effectiveness of the software solution
- Project plan evaluation report (including Gantt chart annotation)

Notes:

- Each of the documents can be in the form of (i) an annotated visual plan or (ii) a written report.
- The submissions correspond to assessment criteria in the school-based assessment rubric.
- Your work must be verified and authenticated by your supervisor.
- The log journal or weblog/blog will provide both supporting evidence that the work is your own and additional material for annotating the Gantt chart.

1 Submission: Client software solution

(Final delivery of your finished software will be negotiated with your teacher.)

Your finished 'product' should be made available for assessment as a compressed (zipped) folder containing a directory of all files required for the software to function with a plain ReadMe.txt file, with instructions for installation and operation.

SCHOOL-ASSESSED TASK TRACKER

☑ Project plan	☑ Justification	☑ Analysis	☑ Folio of alternative designs ideas	☑ Usability tests	☑ Evaluation and assessment	☑ Final submission

9780170440943

Also included in the readMe.txt file would be any passwords to allow entry past security.

Ensure your software solution meets the SRS criteria before submission. This is achieved by continual verification during the PSM – development stage.

2 Document: Testing

This report will contain a comprehensive range of test data that provides:

- testing tables with expected and actual results
- evidence of usability test – how the usability test was designed and delivered
- documentation of results of usability test
- documentation of modifications made to software solution as a result of testing.

Notes:

- You will prepare a usability test that addresses the **core features of your solution**.
- The test must have been undertaken by **at least two other users** and the results recorded.
- You **can make any necessary adjustments to the software solution** based on these results or make suggestions for improvements.

3 Document: Evaluation of efficiency and effectiveness of the software solution

Evaluate the quality of the software solution using criteria from the SRS and Design folio developed for Unit 3.

- Evaluate efficiency and effectiveness of the solution based on the evaluation criteria created in Unit 3.
- Evaluate efficiency and effectiveness of how the software solution meets functional and non-functional requirements.

4 Document: Project plan evaluation report (including Gantt chart annotation)

Evaluate the effectiveness of the project plan and the PSM in developing the software solution. This document will contain:

- annotated project plan with notes and comments explaining which tasks needed to be modified and what the changes were
- your change log, which corresponds with the changes on your project plan
- the evaluation strategies determined earlier to assess the usefulness of the project plan
- the assessment and analysis of the effectiveness of the project plan and usefulness of the project plan.

When completing your project plan evaluation report, consider using headings and dot points where possible. This is not an essay. Refer to the latest VCAA assessment criteria for the report headings.

9780170440943

CHAPTER

6 Cybersecurity risks

KEY KNOWLEDGE

On completion of this chapter, you will be able to demonstrate knowledge of:

Digital systems

- physical and software security controls used to protect software development practices and to protect software and data, including version control, user authentication, encryption and software updates
- software auditing and testing strategies to identify and minimise potential risks
- types of software security and data security vulnerabilities, including data breaches, man-in-the-middle attacks and social engineering, and the strategies to protect against these
- types of web application risks, including cross-site scripting and SQL injections
- managing risks posed by software acquired from third parties.

Data and information

- characteristics of data that has integrity, including accuracy, authenticity, correctness, reasonableness, relevance and timeliness.

Reproduced from the VCE Applied Computing Study Design (2020–2023) © VCAA; used with permission.

FOR THE STUDENT

In this chapter you will focus on the security risks to software and data during the software development process. This chapter introduces the theory and skills necessary for completing Unit 4, Outcome 2. You will also consider the ongoing security risks for the use of the software solution by an organisation. During this chapter you will analyse and evaluate current software development practice security, including the risks to software and data and the consequences of deploying software with ineffective security strategies. You will investigate types of security vulnerabilities, including physical and software controls, web application risks and third-party software risks. Finally, you will develop a risk-management plan to recommend improvements to current practices, after considering an organisation's key legal requirements and ethical issues.

FOR THE TEACHER

This chapter begins consideration of cybersecurity risks during the software development process and deployment of the software solution software and data. Students will analyse and evaluate a range of threats, including physical and software controls to protect software and data including version control, user authentication, encryption and software updates, software auditing, types of software security and data breaches, types of web application risks and, finally, managing risks offered by software through a third party.

By the end of this chapter students should be able to identify and discuss potential risks to software and data security, propose and apply criteria to evaluate the effectiveness of security practices, identify and discuss possible legal and ethical consequences of ineffective security practices for an organisation and to recommend an effective risk-management plan.

The theory and skills for Unit 4, Outcome 2 will be applied to a teacher-provided case study. The assessment task will be determined as either structured questions, a written report or a multimedia report.

Getty Images/Andriy Onufriyenko

9780170440943

Physical and software security controls

It is very difficult today to imagine a world without the Internet, an international network of computer networks that enables instant communication around the world. Web browsers were first introduced around 1994 to provide simple access to web pages containing text, audio and video. The development of the Internet has altered our daily lives in ways that were unimaginable 25 years ago. While users can appear to surf the web without having to identify themselves, attackers can also use this anonymity as a cloak to prevent authorities from finding and prosecuting offenders.

This chapter discusses the threats and vulnerabilities targeted by **malware** and social engineering, and the increasing number of threats targeting applications. Many of these attacks can have immediate impact as they **exploit** a previously unknown vulnerability. These **zero-day attacks** leave no time (zero days) to respond to the threat. There has been no preparation, as the threat or vulnerability was previously unknown. These attacks include web application attacks, client-side attacks and buffer overflow attacks.

Securing the web

Security includes a wide range of considerations. There are many ways and means to get access to information and information systems. An attacker could simply walk in and read data from a screen, or save it to a portable drive. In the online space, access to the administrator privileged root level will allow an online intruder to freely explore all contents of all files. A Chief Information Security Officer (CISO) is a management position responsible for the information security of the entire company. The CISO would devise and implement a strategic plan to prevent, detect and respond to any attempts to breach security.

Security is defined as a combination of:

- confidentiality
- integrity
- availability.

THINK ABOUT SOFTWARE DEVELOPMENT 6.1

Consider Figure 6.1, and Donald Rumsfeld's comments on a lack of evidence to support his view.

1. Identify some known unknowns for cyber security.
2. Are there any unknown unknowns?
3. When does the latter become the former?

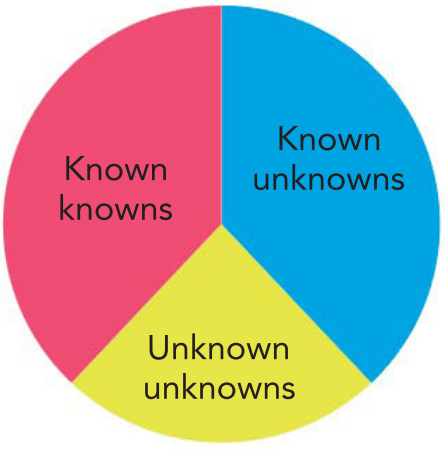


FIGURE 6.1 'There are known knowns. These are things we know that we know. There are known unknowns. That is to say, there are things that we know we don't know. But there are also unknown unknowns. There are things we don't know we don't know ...' – Donald Rumsfeld

9780170440943

Who are the attackers?

While we may never know for sure, we can take a guess at likely types of individuals who may attempt to gain access to information and data or remove assets. These include hackers, spies, insiders, cyber criminals and cyber terrorists.

Hackers

There are several types of hackers. Not all are malicious or are intent on causing disruption and damage. Only two of these actors pose a threat to an organisation.

- White hat hackers, also called ethical hackers, act for good rather than evil purposes. They will offer to repair **virus** damage, or to test for malware to provide a vulnerability alert.
- Grey hat hackers are not malicious, but their **hacking** methods may cross legal or ethical lines.
- Black hat hackers, also called crackers, generate malicious code in several forms intended to steal information and corrupt or disrupt computer operations through unauthorised access to IT systems.
- Script kiddies use downloaded code from a variety of sources – rarely their own. They join the dots by watching 'How to' YouTube videos on how to shape a denial of service (DoS) attack. They use the Internet as their playground, rather than seeking out systems to attack.

Spies

Computer espionage targets specific computer installations, aiming to steal data and information without being detected. Industrial espionage is a common motivator – this is when the aim is to steal company secrets and provide them to competitors.

Insiders

The most effective attack on an organisation comes from an unlikely source known as an insider. Employees, sub-contractors and business associates can abuse their access to company information and systems. Many attacks are simple sabotage or theft of intellectual property, although carelessness also contributes to data breaches. For example, hundreds of notebook computers are lost or misplaced every day. These portable computers often contain sensitive commercial information, personal details, client lists, price lists and company strategies. Departing employees often 'take their work' with them to the new place of employment. Company policy on staff departures needs to be clearly articulated.

Cybercriminals

Hacking is big business. **Cybercriminals** or skilled hackers establish schemes that harvest information and scam unsuspecting computer users or legitimate companies, from large corporations to small family businesses and individuals. For example, some black hat organisations operate call centres and run phone scams. In one such scam, the hacker rings a person and claims to work for Microsoft, and offers to help with a computer problem. After the hacker convinces the potential victim to grant remote access to their computer or to download and install software, the criminal harvests passwords and banking information or takes over the computer and uses it to launch attacks on others. The victim is often charged a fee for this 'help'.

WannaCry **ransomware** was released in May 2017. Within two weeks an estimated 400 000 computers in 15 countries were infected. Security experts released decryption tools quickly to limit the damage to approximately US$150 000.

SCHOOL-ASSESSED TASK TRACKER

☑ Project plan	☑ Justification	☑ Analysis	☑ Folio of alternative designs ideas	☑ Usability tests	☑ Evaluation and assessment	☑ Final submission

9780170440943

Sophisticated criminal organisations set up hundreds of schemes to defraud users. One such scheme exploits so-called celebrity influencers. 'Likes', followers and friend requests can be bought online to provide instant popularity. Fake Facebook and Twitter accounts are easily automated to generate thousands of likes a minute. Twitter has a follower limit of 1000 per day, and a total limit of 5000. Twitter has 260 million accounts with 46 per cent of these active on the platform daily.

Cyberterrorists

Terrorists are motivated by an ideological belief, and will attack on the basis of their principles. Cyberterrorists are considered unpredictable, as the frequency and timing of their attacks are strategically chosen to maximise disruption to computer users, and to interrupt systems and networks that result in infrastructure outages or corruption of community data. For example, they might target computers that control the electricity power grid, traffic control systems or railway or airport control systems.

What are the likely threats?

While the essence of the Rumsfeld statement referred to in Figure 6.1 may still apply, there are many 'known known' threats. Threats represent a constant danger to an asset, and while the details will vary according to the specific situation, threats have been well researched and are well understood.

There are several categories of threat and they require different responses by those entrusted to ensure security of data, information assets and personnel.

TABLE 6.1 Categories of threats

Category of threat	Examples
Acts of human error	Accidents, incorrect assumptions, inadequate training, inexperience, ignorance of policy requirements, leaving a back door open
Intellectual property compromised	Piracy, copyright infringements
Espionage	Unauthorised access and data collection
Extortion	Blackmail where ransoms are demanded in return for release of stolen data or control of systems
Sabotage or vandalism	Destruction of systems or information
Theft	Illegally taking data, physical assets or IP
Software attacks, including malware	Viruses, **worms**, macros, XSS, SQL, XML injections, denial-of-service
Forces of nature	Fire, flood, earthquake, lightning
Technical hardware failure or errors	Equipment failure, known equipment weaknesses
Technical software failure or errors	Bugs, code problems, unknown loopholes, known loopholes (back doors)
Technological obsolescence	Outdated technology causes the system to be unreliable and untrustworthy

9780170440943

Data security

The purpose of any information system is to provide reliable and predictable performance. To ensure an IT system is performing as expected, specific standards and criteria have been devised through collaboration with industry and government.

Trust and confidence in the quality of the information within a system is compromised if any of the three components shown in Figure 6.2 is violated.

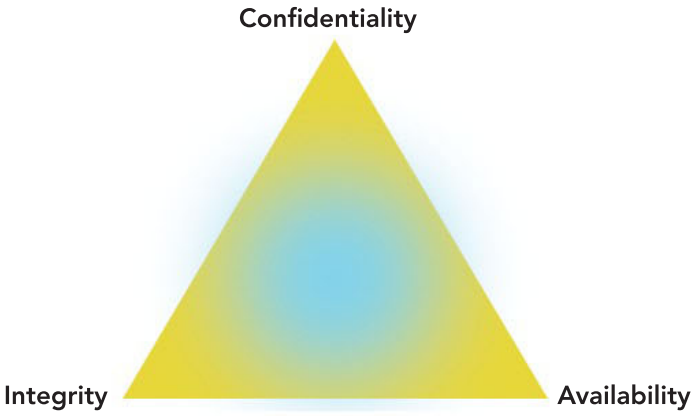


FIGURE 6.2 The concepts of confidentiality, integrity and availability (CIA) in relation to data security were devised over many years as use of computer systems increased to become commonplace, widespread and ultimately ubiquitous.

Confidentiality

Access to sensitive information is available only to those who have authorisation; anyone not authorised is denied access. Implementation relies on usernames, passwords, access control lists and encryption. Permission may be categorised on several levels according to the sensitivity or amount of impact or damage that may be done if the information should get into unauthorised hands. Security clearance levels of access used by military organisations are sometimes adopted. This requires material to be classified into six levels of access:

- top secret
- secret
- confidential
- restricted
- official
- unclassified.

If a person holds a security clearance, they may be eligible to view a category of material; however, there is still a requirement that they must 'need to know'.

The Australian Government Security Classification system has four levels:
- top secret
- secret
- confidential
- protected.

A dissemination limiting markers scheme further limits disclosure by declaring material to be For Official Use Only (FOUO) or Sensitive.

SCHOOL-ASSESSED TASK TRACKER

☑ Project plan	☑ Justification	☑ Analysis	☑ Folio of alternative designs ideas	☑ Usability tests	☑ Evaluation and assessment	☑ Final submission

9780170440943

Integrity

Information must be consistent with the original purposes and have true and correct details. The receiver of the information must be able to trust the source and be assured that any edit or modification has been performed only by those authorised. Integrity can be assured when data and information is stored using security mechanisms such as encryption or hashing. Backups that are identical to the source must also be kept.

Availability

Access is available to those permitted to use the materials. Hardware is maintained in a ready state so that issues of hardware maintenance and redundancy, software updates, patching and network access issues do not affect the user. Dedicated hardware can overcome the serious consequences of downtime, or when denial of service (DoS) attacks occur.

Physical security

© Commonwealth of Australia 2019. Australian Government: Attorney-General's Department

FIGURE 6.3 The Australian Government Protective Security Policy Framework describes the physical protections required to safeguard people, information and assets.

While there are many interpretations of processes to safeguard the physical security of people and assets, the Australian Government has a fully documented protocol readily available online. The Protective Security Policy Framework (PSPF) specifies in exhaustive detail the steps required to minimise or remove security risks to people, information and assets.

The core requirement states:

Each entity must implement physical security measures that minimise or remove the risk of:

a harm to people

b information and physical asset resources being made inoperable or inaccessible, or being accessed, used or removed without appropriate authorisation.

In summary:

… that entities protect their resources by using a combination of physical and procedural security measures to achieve this outcome. These include measures to:

a Deter – measures that cause significant difficulty or require specialist knowledge and tools for adversaries to defeat

b Detect – measures that identify unauthorised action is being taken or has already occurred

c Delay – measures to impede an adversary during attempted entry or attack, or slow the progress of a detrimental event to allow a response

d Respond – measures that prevent, resist or mitigate an attack or event when it is detected

e Recover – measures to restore operations to normal levels (as soon as possible) following an event.

Protective Security Policy Framework (PSPF)

© Commonwealth of Australia 2019. Australian Government: Attorney-General's Department. Released under CC BY 4.0 International, link to license: https://creativecommons.org/licenses/by/4.0/

9780170440943

Requirement 1. Design and modify facilities	When designing or modifying facilities, entities **must**: i. secure and control access to facilities to meet the highest risk level to entity resources ii. define restricted access areas as detailed below.	
	Zone name	**Zone definition**
	Zone One	Public access.
	Zone Two	Restricted public access. Unrestricted access for authorised personnel. May use single factor authentication for access control.
	Zone Three	No public access. Visitor access only for visitors with a need to know and with close escort. Restricted access for authorised personnel. Single factor authentication for access control.
	Zone Four	No public access. Visitor access only for visitors with a need to know and with close escort. Restricted access for authorised personnel with appropriate security clearance. Single factor authentication for access control.
	Zone Five	No public access. Visitor access only for visitors with a need to know and with close escort. Restricted access for authorised personnel with appropriate security clearance. Dual factor authentication for access control.

© Commonwealth of Australia 2019. Australian Government: Attorney-General's Department. Released under CC BY 4.0 International, link to license: https://creativecommons.org/licenses/by/4.0/

FIGURE 6.4 PSPF indicative zone names and definitions for physical access

Layering zones

Entities are advised to layer zones, working in from Zone One (public access areas), and increasing the level of protection with each new zone. Multiple layers are the 'delay' design feature, as they provide more time to detect unauthorised entry and respond before resources are compromised.

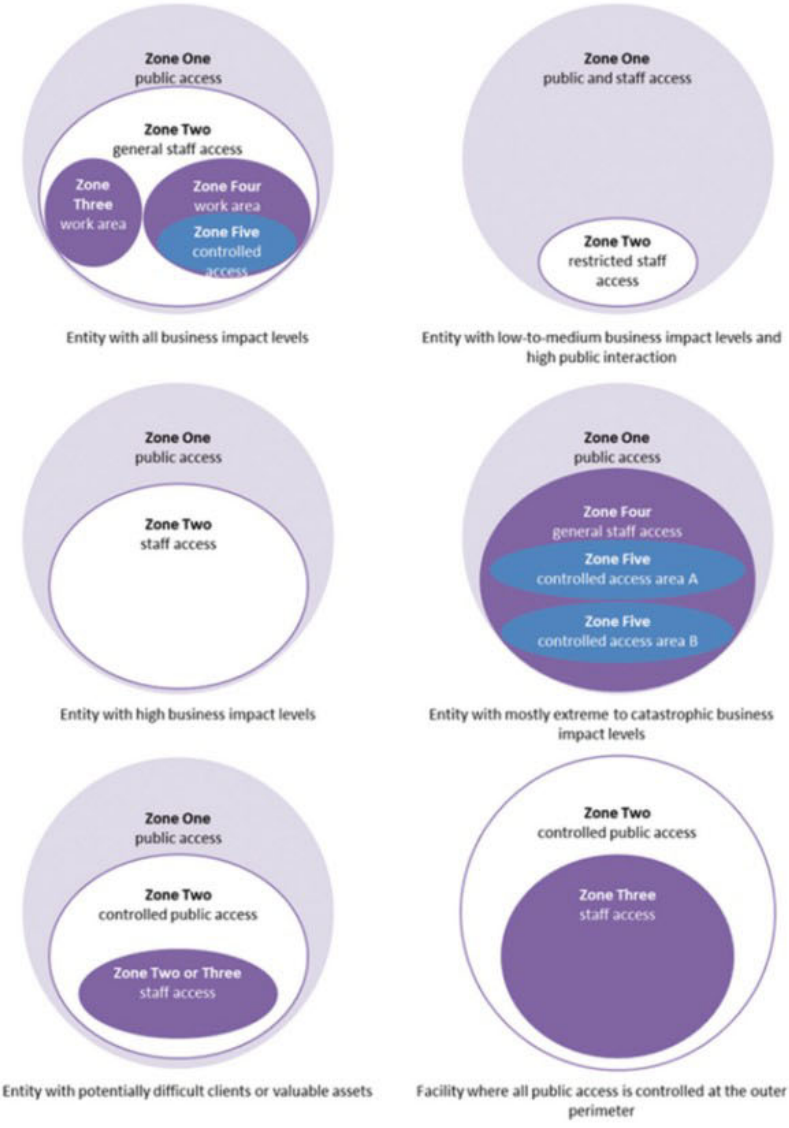


FIGURE 6.5 These diagrams demonstrate indicative layering of zones implemented for different locations. In some instances it may not be possible for higher zones to be fully located within lower zones, and so higher zone areas may need to be strengthened.

© Commonwealth of Australia 2019. Australian Government: Attorney-General's Department. Released under a CC BY 4.0 International license, link to license: https://creativecommons.org/licenses/by/4.0/

SCHOOL-ASSESSED TASK TRACKER

☑ Project plan	☑ Justification	☑ Analysis	☑ Folio of alternative designs ideas	☑ Usability tests	☑ Evaluation and assessment	☑ Final submission

9780170440943

A secure building with IT resources

A typical building with sensitive data resources and a high risk of losses if the IT systems were compromised would employ a variety of security measures to deter, detect or delay an intruder. These include:

- closed circuit television cameras (CCTV) to warn guards of approaching people, and to provide a record identifying individuals if evidence is later required
- motion sensors to activate CCTV monitoring and security lighting to record all activity in the immediate area
- physical barriers to isolate areas of the building
- security passes to authorise access to secure areas
- biometric authentication for authorised personnel
- alarm systems, such as perimeter (or external) intrusion detection systems (PIDS) or alarms and internal security alarm systems
- security guards, which provide the highest level of scrutiny when available 24/7 with random interval patrols every four hours.

THINK ABOUT SOFTWARE DEVELOPMENT 6.2

Consider the diagram of the typical secure building shown in Figure 6.6.

Identify which secure measures are to:

a deter

b detect

c delay.

In addition, are there any other measures that could be deployed?

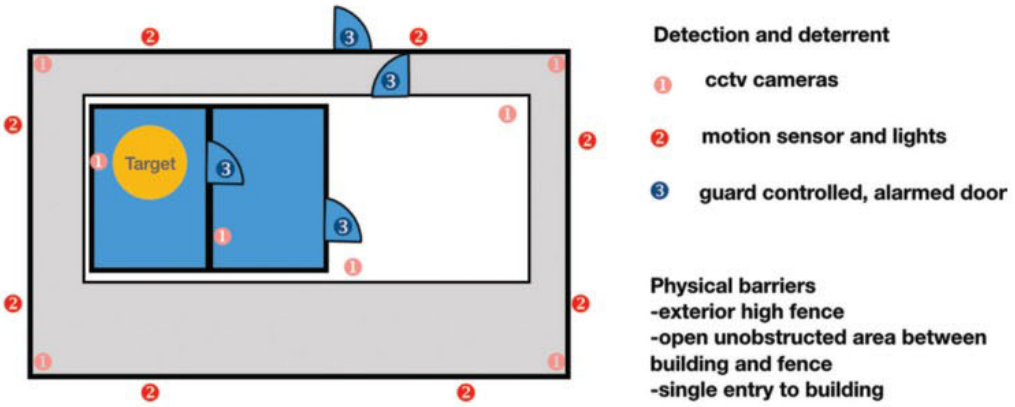


FIGURE 6.6 A typical secure building layout

A typical office environment

There are many opportunities to improve security of assets and information in a typical office environment. The entire network and all its components need physical protection to ensure confidentiality, integrity and availability.

Lock the server room

Keep the server room door locked at all times, and ensure the automatic door closer has a short return time to deter **tailgating**, or **piggyback entry**. Physical locks are the single best deterrent once an intruder has breached all other barriers. Master keys and keycodes need to be recorded to identify who has authorised access. Electronic locks record both time of access and the user code.

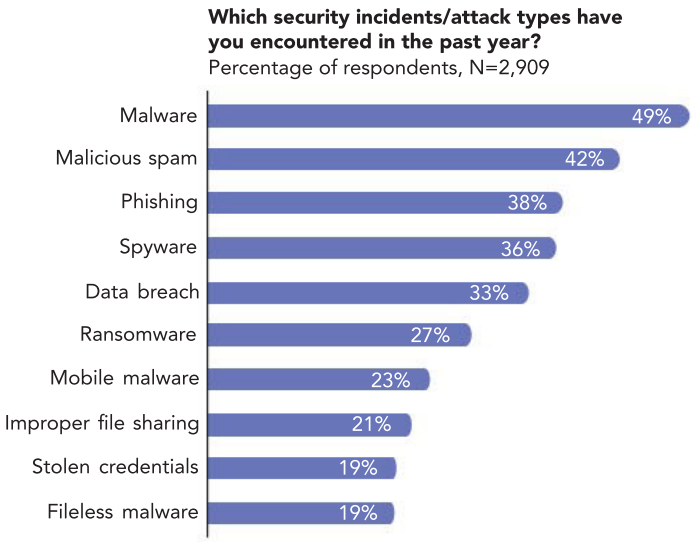
Traditional keylocks require CCTV monitoring to identify who enters and when.

Software security

Over time, CISOs and other IT managers have become aware that there is no single way of securing information.

To develop secure applications, begin by assuming that all data received by the application is from an untrusted source. This applies to all data received – data, cookies, emails, files or images. This includes users who have logged in to their account and authenticated themselves. There is no preferential treatment to any user; all should be subject to the entire range of security measures.

Not trusting user input means always validating it for type, length, format and range. The data is challenged each time it is processed; for example, when data is entered through a web form to an application script, and encoded before being displayed on a dynamic page. Logistically, this means that any values accepted from the client side are checked, filtered and encoded before being passed back to the user. Any user-supplied data handling and processing must be certified as secure.



Based on data from Cisco's Cybersecurity Series 2019: 'Anticipating the Unknowns: Chief Information Security Officer (CISO) Benchmark Study', March 2019.

FIGURE 6.7 A survey across many businesses and several countries identified the most frequent risk of attacks. Notably, the first three all use email to gain access to a computing device.

Security attacks

Billions of dollars and thousands of hours are expended each year on computer security. The number of attacks that are prevented increases each year, as do the number of successful attacks.

How do these attacks occur?

Recent attacks include malware and online banking attacks.

1 Malware: Fake anti-virus attacks are responsible for the majority of the malware delivered by web advertising. For example, a user clicks on an advertisement on a web page offering a free virus scanner. Suddenly a warning declares that the computer is infected. A pop-up window invites the user to purchase anti-virus software to clean their computer. At this point the window cannot be closed, and rebooting does not clear the pop-up window. Many users enter their credit card details to purchase the software; these are transmitted to the attacker, who uses them to make online purchases. Malware often remains on a computer until the computer drive is wiped and a fresh operating system is installed.

THINK ABOUT SOFTWARE DEVELOPMENT 6.3

The Australian Government will spend $38.7 billion on national security in the 2019–2020 financial year. Some of these funds are allocated to defending against cyberattacks as part of the government's CyberSecurity Strategy. Quick responses are required in the event of a targeted attack against government IT systems.

The Australian Cyber Security Centre delivers intelligence, cyber security and offensive operations in support of the Australian Government and Australian Defence Force (ADF).

1 Use the weblink below to research the details of the Cyber Security Strategy.
2 What are the cyber security goals?
3 How might these goals affect you, either directly or indirectly?

Australia's Cyber Security Strategy

SCHOOL-ASSESSED TASK TRACKER

Project plan	Justification	Analysis	Folio of alternative designs ideas	Usability tests	Evaluation and assessment	Final submission

9780170440943

In June 2016, 94 per cent of adult Australians used the Internet to conduct banking, pay bills, or buy and/or sell goods and services.
Source: Cybersecurity Strategy, First Annual update 2017, Australian Government

2 Online banking: The increase in the use of mobile devices, smartphones and tablets has brought a corresponding increase in 'convenience apps'. These applications remove the 'friction' when interacting with certain activities. For example, one-touch purchasing and fingerprint or facial recognition remove the delays in purchasing online products and services. If a consumer can be convinced (i.e. 'conned'), then the purchase is approved, often without goods and services being supplied, or with exorbitant shipping fees and delays. Stolen credentials and payment authority are increasingly common, with unsecured wi-fi transmitting usernames, passwords and credit card details unencrypted. Shopping centre wi-fi is often imitated by the attacker. The goal is often to capture online banking details transmitted across wi-fi networks, rather than a single transaction with a credit card. For example: paying for car parking by smartphone on wi-fi, instead of via the more secure 4G network.

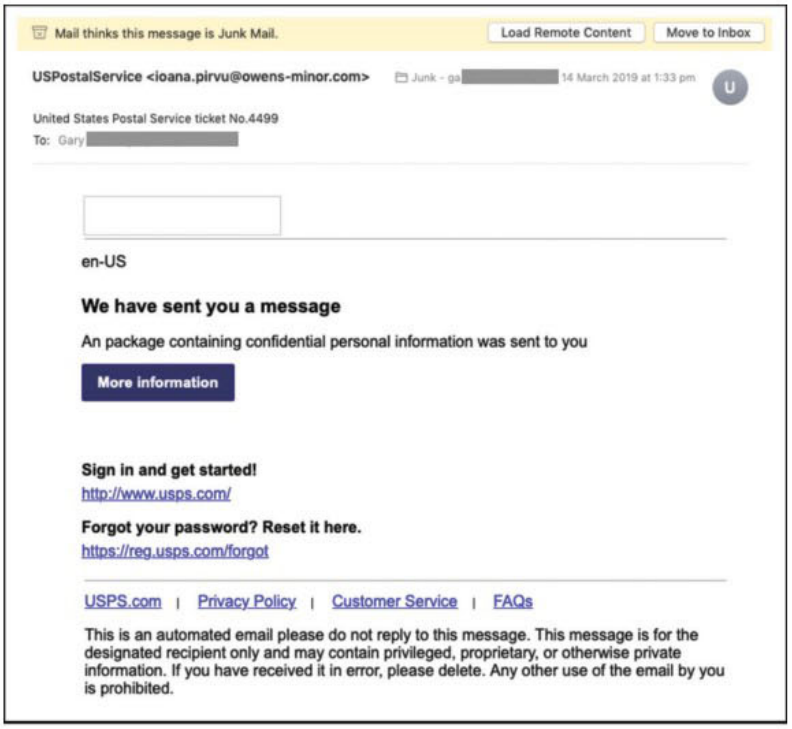


FIGURE 6.8 Phishing for your details. Spelling and grammar errors are the first indication of a scam. The unusual domain name in the email should also increase concerns.

ISSUE

Cryptojacking malware increases 4000% in 2018

Crypto mining scams occur when a hacker accesses someone's computer without their permission to mine for digital tokens. Hackers will infiltrate a computer by tricking a person into clicking on a malicious link in an email or they will infect a website to gain access.

The rise of crypto mining malware this year has displaced ransomware, which was a huge story in 2017 as bitcoin and other digital token prices plummeted. With hackers no longer making money off holding individuals' and companies' data for ransom, they searched for new ways to earn a living. Crypto mining malware is attractive because it can go undetected for a very long time, but it takes longer to make money off the scam. A ransomware attack can yield a hacker more money in less time, but the victim will become aware very soon and could baulk at paying.

Source: From Forbes.com © 2018 Forbes. All rights reserved. Used under license.

Other technical threats that seek to exploit a vulnerability are listed in Table 6.2.

TABLE 6.2 Technical threats

Threat	Description
Back door	Program feature left by developers during maintenance or installed by malware
Brute force	By sheer computing power, every combination of a password is tested
DoS and DDoS	Overwhelming number of connection requests, which effectively block access to the website
Password dictionary	Using known passwords from data breaches reduces the number of guesses to break in compared with random or brute force attacks
DNS cache poisoning	DNS data is replaced with bogus URL and redirects to an attacker's fake facsimile website
Hoax	Time wasted on fake threats or attacks
Mail bombing	Huge volumes of email directed at a target
Malicious code or malware	Viruses, worms, trojans, logic bombs, active scripts C++, PHP, Python, Ruby, VBnet, Flash, SQL injection, cross-site-scripting
Man-in-the-middle	A network connection is hijacked, enabling a third party to see and modify the transferred data
Rootkit	Sections of the computer operating system are altered or replaced to allow prohibited activity
Sniffer	Data over a network is monitored by software or a device
Spear phishing	Targeted social engineering phishing invitations exploit personal and financial information, particularly through email and social media. Increasingly focused on known wealthy individuals.
Spoofing	Unauthorised access is gained by fooling the host computer into trusting a fake IP address

Malware

There is a huge variety of malicious software that can take advantage of weaknesses in the host computer software or operating system. The main objective of viruses and worms is to spread infection, while others aim to remain concealed and undetected. All, however, enter the computer system without the user's knowledge and perform unwanted and sometimes harmful actions.

Viruses

A computer virus is malicious code that reproduces itself on the same computer. The code inserts itself into files and modifies, corrupts or destroys the ability of the file to operate as normal. There are many ways the virus can infect the data file or program:

a Appender infection

b Swiss cheese infection

c Split infection

9780170440943

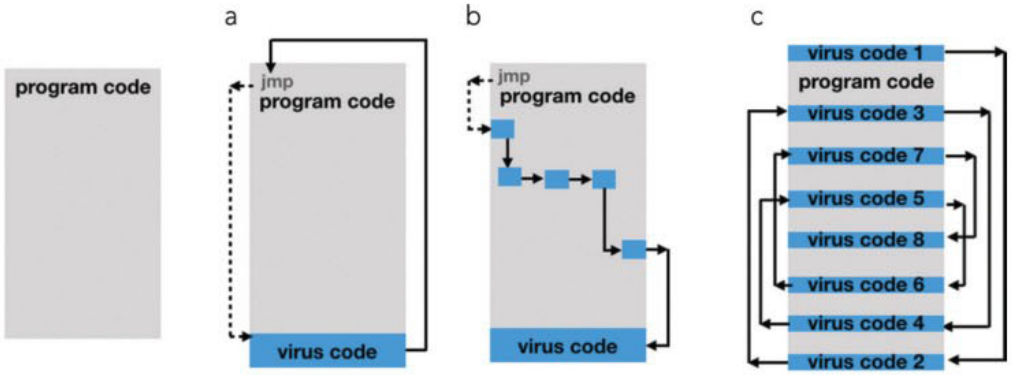


FIGURE 6.9 Types of virus infection that can overwhelm a file collection on local storage media: **a** Appender virus; **b** Swiss cheese virus; **c** Split infection virus

Worms

A worm is a self-replicating malicious code that can spread across computer systems, which can modify, corrupt or delete data or information. Other programs can also be run undetected by the legitimate user and operated remotely as 'zombie bots'.

Trojans

This code is named after the story from Greek mythology where the Trojan Horse appears to be one thing, but is actually another. **Trojans** often take the form of 'free virus-checking software', when in fact they are malware and can be virus propagators. Permission must be granted to allow of the software to be installed on the computer. The user must be convinced to enter the appropriate password to activate installation.

This attack is difficult to counter, as the weakness is the human user who has administrative permission to install software. If that person installs the software, the computer can be infected. Preventative measures are mostly defensive, such as maintaining full incremental backups of system and user files and applying anti-virus software that scans for known characteristics. The disadvantage with both of these strategies is that the newer malware will not be detected and the computer hard drive will need to be wiped and a full system reinstalled.

Logic bomb

Malicious software will lie dormant until a timer or conditional trigger goes off, which will activate at a pre-determined time or when certain circumstances are satisfied. Logic bombs are placed by programmers or developers who have access to the source code.

Detection and prevention

In Australia, the Australian Competition & Consumer Commission (ACCC) maintains a Scamwatch website, where instances of the frequency and estimated costs of various scams are presented.

The Australian Cyber Security Centre (ACSC) provides:

- monitoring of cyber threats across the globe 24 hours a day, seven days a week so that Australians can be alerted early on about what to do
- advice and information about how to protect individuals and businesses online

THINK ABOUT SOFTWARE DEVELOPMENT 6.4

Explore the latest Scamwatch statistics at the ACCC website.

1. What are the limitations on the information displayed?
2. If these statistics are an estimate, will the actual numbers be higher or lower than those displayed? Explain your reasoning.
3. For the current year, which category has:
   - **a** the most reported scam?
   - **b** the greatest amount lost?
   - **c** the most frequent delivery method?
   - **d** the most affected age group?
4. Why is the connection between 'Amount lost' and 'Number of reports' inconsistent?

Scamwatch

© Commonwealth of Australia 2019. Australian Government: Attorney-General's Department. Released under CC BY 3.0 AU, link to license: https://creativecommons.org/licenses/by/3.0/au/deed.en

FIGURE 6.10 ACCC Scam statistics for 2018

- clear and timely advice to individuals, small to medium business, big business and critical infrastructure operators when there is a cybersecurity incident
- investigation and solution development to cybersecurity threats
- services to fight **cybercrime**.

Security procedures related to malicious software are shown in Table 6.3.

© Commonwealth of Australia 2019

FIGURE 6.11 The Australian Cyber Security Centre is part of the Australian Signals Directorate, which works in matters of intelligence, cybersecurity and offensive operations.

SCHOOL-ASSESSED TASK TRACKER

Project plan	Justification	Analysis	Folio of alternative designs ideas	Usability tests	Evaluation and assessment	Final submission

9780170440943

TABLE 6.3 Security procedures for malicious software or malware

Detection	Prevention equipment	Prevention policies	Countermeasures
Malicious software (malware) may cause: Changes in file sizes or date/time stamps Computer slow starting or slow running Unexpected or frequent system failures Change of system date/time Low computer memory or increased bad blocks on disks Unexpected data loss or file corruption Performing a virus scan may uncover potential threats	**Firewall**: A firewall is a hardware device or software that blocks unauthorised access from particular locations or to particular parts of a computer. For example, web traffic is normally sent and received on port 80 of your modem; a firewall can be configured to block access from all other ports. Firewalls are very good at stopping the spread of 'worms' but ineffective with other types of 'malware'. A network firewall resides between two networks (usually the Internet and the organisation's network). A client firewall is software that runs on the end user's computer. **Antivirus software:** Antivirus software can scan your system for known viruses or detect system behaviour that may indicate an infection. **Spam filter:** Worms often present themselves as an email attachment. Having an effective spam filter with up-to-date definitions can prevent this.	Limit user privileges. Limiting user privileges will limit the damage that can be done. Limit user connectivity and user downloads. Limit executable files and limit media that can be used for loading data and software. Provide thorough user training. Many threats, including trojans, rely on inadvertent user cooperation. User training is the best prevention. Perform security upgrades. Upgrade via security patches as soon as they become available. Keep all virus and spam definition files up to date. Perform regular backups. In the event that infection cannot be prevented, the data needs to be able to be efficiently restored.	Quarantine affected equipment and remove from the network. Infected computers may need to be completely reformatted and all data destroyed. Run antivirus software and attempt to locate and destroy known viruses. Determine source of infection and issue alert. Depending on the nature of the threat, the alert may need to be issued not only to other system admins, but to all network users. Restore any lost data with most recent backups and replace any damaged equipment. Enact business continuity plan in order to limit the amount of downtime and minimise the negative consequences of the infection.

Software development practices

The software development life cycle (SDLC) may take months or years before a product is released. The product is then maintained through various updates and eventually retired. Over that entire period of time, the product is expected to withstand many potentially hostile acts and still preserve the integrity, confidentiality and availability of its underlying functions, database and customer/consumer information.

See chapter 5 for further details about Waterfall and Agile SDLCs.

The process of securing the integrity of the software begins in the design and development stages. In a traditional 'waterfall', SDLC security might come towards the end of each stage in the development cycle. In recent times, 'agile' SDLC methods have integrated security considerations into early discussions about how the software operates and identifying essential functions.

Frequently, a security software development life cycle (secSDLC) is necessary to apply known solutions and preventative measures to the developing product.

Often, vulnerabilities are caused unintentionally. Software developers insert sections of code as a temporary measure, with every intention of returning to 'fix' this short-term measure. Too frequently these insecure methods are used in order to meet a deadline. Project management and version control software has removed the opportunity for insecure code that is 'not ready' or does not meet production requirements. The 'new' strategy does not allow the creation of a potential weakness or vulnerability. Instead, only compliant code can be added to the project; any code that is not yet ready is simply not permitted to be added to the project. (See the case studies on ATLASSIAN project tracking on page 225, and Apache Subversion on page 226.)

9780170440943

How to protect software and data

Data must be protected in line with legal requirements (see chapter 7 for legislation implications) during the stages of development and after release of the product. Frequently, 'real data' may be used for testing purposes, when protections may not have been put in place.

Software may be a 'work in progress', but it can be an attractive target for those interested in gaining commercial advantage, early release for bootleg copies, or with other motivations. Software security in this instance may be more vulnerable and more important than after the release of the gold master final version.

There may be large numbers of employees with proximity and access to the master file storage. The security protocols outlined in the security plan will specify when and who has access, and this needs to be consistently applied during the development and release stages.

Version control

As all software developers quickly discover, keeping track of changes and the working up-to-date 'master' file is a challenging task. Frequently, soon after changes are made, a reversion to the previous unmodified file is necessary. Should the developer choose to keep one master file, all changes will not be recorded. Clearly a log of all changes is necessary. This **version control** records each stage of the development so that it is possible to 'roll back' to any previous point in the development.

When teams of developers work on several streams of the software, versions and stages become important, and the issue of merging the improvements becomes a new challenge. Software control of versions has been developed with 'Subversion', a popular open source option. Atlassian provides a commercial ecosystem for software development, and is free for teams of less than five. Version control may also be integrated into the programming language IDE.

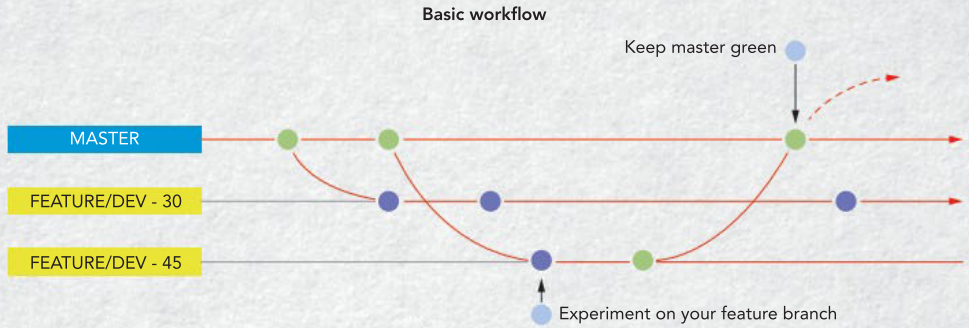
CASE STUDY

Atlassian

Creating a branch for each issue makes it easy to hand-pick which changes to ship out to production or bundle into a release. Since you are not dog-piling changes onto master, you get to select what comes into master – and when. You can ship an epic's MVP plus one nice-to-have, rather than wait until all the nice-to-haves are fully baked. Or ship a single bug fix and do it within the framework of a regular ol' release. Even if the fix is urgent, you won't have to deal with the three-ring circus of backing out other changes that aren't ready to ship yet just to get that one change out the door.

And that ease of shipping a single code change is the essence of continuous delivery.

Text extract from 'Super-powered continuous delivery with Git', by Sarah Goff-Dupont, accessed from https://es.atlassian.com/continuous-delivery/principles/why-git-and-continuous-delivery-are-super-powered



© Atlassian, link to article: https://www.atlassian.com/continuous-delivery/principles/why-git-and-continuous-delivery-are-super-powered

FIGURE 6.12 Atlassian software creates a development branch for each story or bug-fix or task you implement.

SCHOOL-ASSESSED TASK TRACKER

☑ Project plan	☑ Justification	☑ Analysis	☑ Folio of alternative designs ideas	☑ Usability tests	☑ Evaluation and assessment	☑ Final submission

9780170440943

CASE STUDY

Apache Subversion

Apache Subversion (SVN) is a system of software versioning and revision control. Subversion was created by CollabNet Inc. in 2000, and is now a top-level Apache project being built and used by a global community of contributors.

Subversion uses branches and tagging to map versions. A branch represents a separate line of development. Tagging refers to labelling the repository at a certain point in time so that it can be easily found in the future. In Subversion, the only difference between branches and tags is how they are used.

A new branch or tag is set up by using the 'svn copy' command, which should be used in place of the native operating system mechanism. The copied directory is linked to the original in the repository to preserve its history, and the copy takes very little extra space in the repository.

All the versions in each branch maintain the history of the file up to the point of the copy, plus any changes made since. One can 'merge' changes back into the trunk or between branches.

Subversion is distributed as open source under the Apache License. Subversion is used by software developers to maintain current and historical versions of files such as source code, web pages and documentation. Its goal is to be a mostly compatible successor to the widely used Concurrent Versions System (CVS). The open source community has used Subversion widely in projects such as Apache Software Foundation, FreePascal, FreeBSD, GCC and SourceForge.

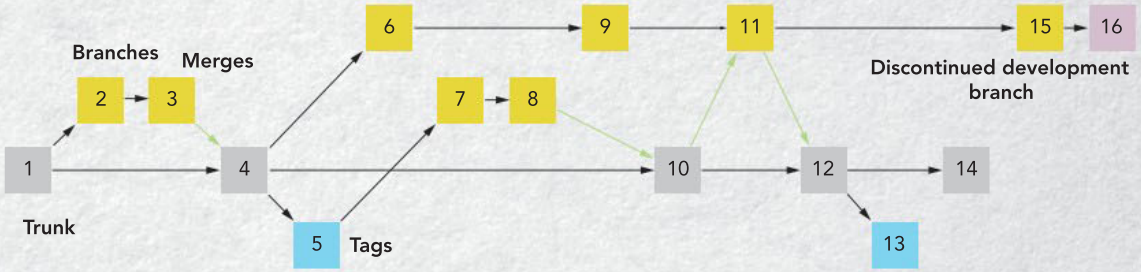


FIGURE 6.13 A project plan using Subversion

THINK ABOUT SOFTWARE DEVELOPMENT 6.5

A common developer saying is: 'Beware of code written by just one person'.

Most software is developed by teams. Sometimes hundreds of people are involved in the various stages of construction, exploration and production following an Agile method of SDLC. Provide at least two reasons to explain why this might be.

User authentication

The ACSC recommends using certain techniques to prevent cybersecurity threats. These techniques include clearly documenting and training employees in cybersecurity systems and plans, and designing and implementing cybersecurity awareness programs for all employees.

To prevent or lessen the impact of data spills and breaches and other cybersecurity incidents, the ACSC advises the implementation of the following steps.

1. Require all users to periodically reset passwords to reduce the ongoing risk of credential compromises.
2. Consider increasing password length and complexity requirements to mitigate the risk of brute-force attacks being successful.
3. Implement a lockout for multiple failed login attempts.
4. If credentials have been compromised, reset passwords as soon as possible.
5. Discourage users from reusing the same password across critical services such as banking and social media sites, or sharing passwords for a critical service with a non-critical service.
6. Recommend the use of passphrases that are not based on simple dictionary words or a combination of personal information. This reduces the risk of password guessing and simple brute-forcing.

9780170440943

7 Advise users to ensure new passwords do not follow a recognisable pattern. This reduces the risk of intelligent brute-forcing based on previously stolen credentials.
8 Use multi-factor authentication for all remote access to business systems and for all users when they perform a privileged action or access an important (sensitive/high-availability) data repository.
9 Look out for unusual account activity or suspicious logins. This may help detect when a service such as email has been compromised and needs a password reset.
10 Encourage users to think carefully before entering credentials.
   - Ask if this is normal.
   - Don't enter credentials into a form loaded from a link sent in email, chat or other means open to receiving communications from an unknown party.
   - Even if the page looks like the service being reset, think twice.
   - Do not click the link. Instead, browse to the website and reset the password from there.
   - Be aware that friends' or other contacts' accounts could be compromised and controlled by a third party to also send a link.
11 If some credentials have been compromised, try to identify a specific cause. Were the credentials entered in an untrusted place? Were they recently reset? What were the credentials for? Were the credentials used elsewhere?
12 Keep operating systems, browsers and plugins up-to-date with patches and fixes.
13 Enable anti-virus protections to help guard against malware that steals credentials.

© Office of the Australian Information Commissioner — www.oaic.gov.au. Extract from 'Information from the Australian Cyber Security Centre about preventing and mitigating data breaches'. Released under CC BY 3.0 AU, link to license: https://creativecommons.org/licenses/by/3.0/au/deed.en

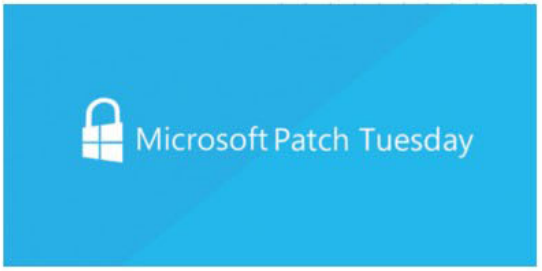
Encryption

The encryption strategy recognises that data will be exposed to unauthorised access, so the data is rendered useless by encoding it to make it unintelligible to all but those properly authorised and accredited. There are several forms of encryption readily available. Pretty Good Privacy or PGP has been available since 1991.

Software updates

Vulnerabilities to existing systems are constantly identified and a fix is created by the distributors of the software. These fixed versions are released by software companies as a 'patch' to be inserted in the operational software wherever it may be installed. The obligation for installing and updating the vulnerable software is entirely on the user or operator of the software. The software company has a responsibility to make the patch available, but it is up to the software operator to update and ensure the software and system is as secure as it can be. (See the WannaCry case study, page 228.)

There are dire implications for running compromised systems. For example, compensation is likely to be due to affected users of the system if all reasonable steps have not been taken to secure users' personal data and information.



Used with permission from Microsoft

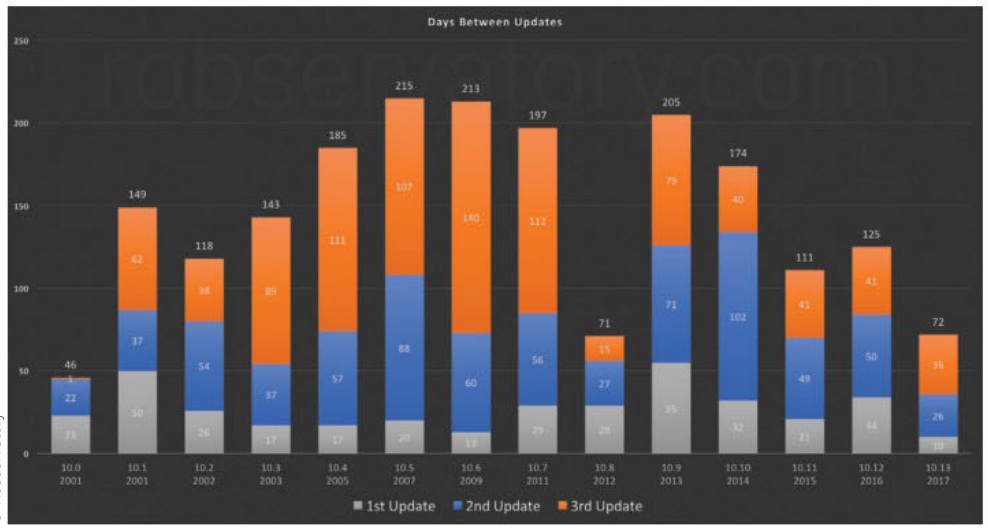
FIGURE 6.14 'Patch Tuesday' is when Microsoft releases updates and patches every second Tuesday.

SCHOOL-ASSESSED TASK TRACKER

☑ Project plan	☑ Justification	☑ Analysis	☑ Folio of alternative designs ideas	☑ Usability tests	☑ Evaluation and assessment	☑ Final submission

9780170440943

Software updates are a regular step in the maintenance of deployed software products. Operating systems are frequently updated according to a published timetable or as the need arises.



© Robservatory

FIGURE 6.15 macOS updates for OSX. There is a major release each year; dot updates have been irregular or as the need demands.

CASE STUDY

WannaCry is a ransomware cryptoworm

In 2017 the WannaCry cryptoworm, also known as WannaCrypt, targeted Microsoft Windows operating systems by encrypting data and demanding ransom payments in Bitcoin cryptocurrency.

The worm automatically spread itself by using a vulnerability that was first identified by the US National Security Agency but was withheld to allow the exploit to be used for NSA purposes. WannaCry has been estimated to affect 230 000 computers in 15 countries over a period of two weeks.

Ninety-eight per cent of the affected systems were running Windows 7 and 8, with the remainder running Windows XP. Microsoft had released a patch several months earlier that would have prevented the attack, but only if Windows updates were enabled. The affected computers had not received an update since the last patch issued by Microsoft in 2014.

To avoid such exploit attacks:

1. Maintain the operating system by installing all security updates immediately. Microsoft issued a patch in March; WannaCry did the damage in May.
2. Only run supported operating systems. Both Windows XP and Windows 7 are no longer receiving updates.
3. Software developers rely upon notification of vulnerabilities as and when they are found. The vulnerability was known but had been stockpiled, rather than disclosed.

Roke via Wikimedia. Released under CC BY SA 3.0, link to license: https://creativecommons.org/licenses/by-sa/3.0/deed.en

FIGURE 6.16 Widespread impact of the WannaCry cryptoworm attack

9780170440943

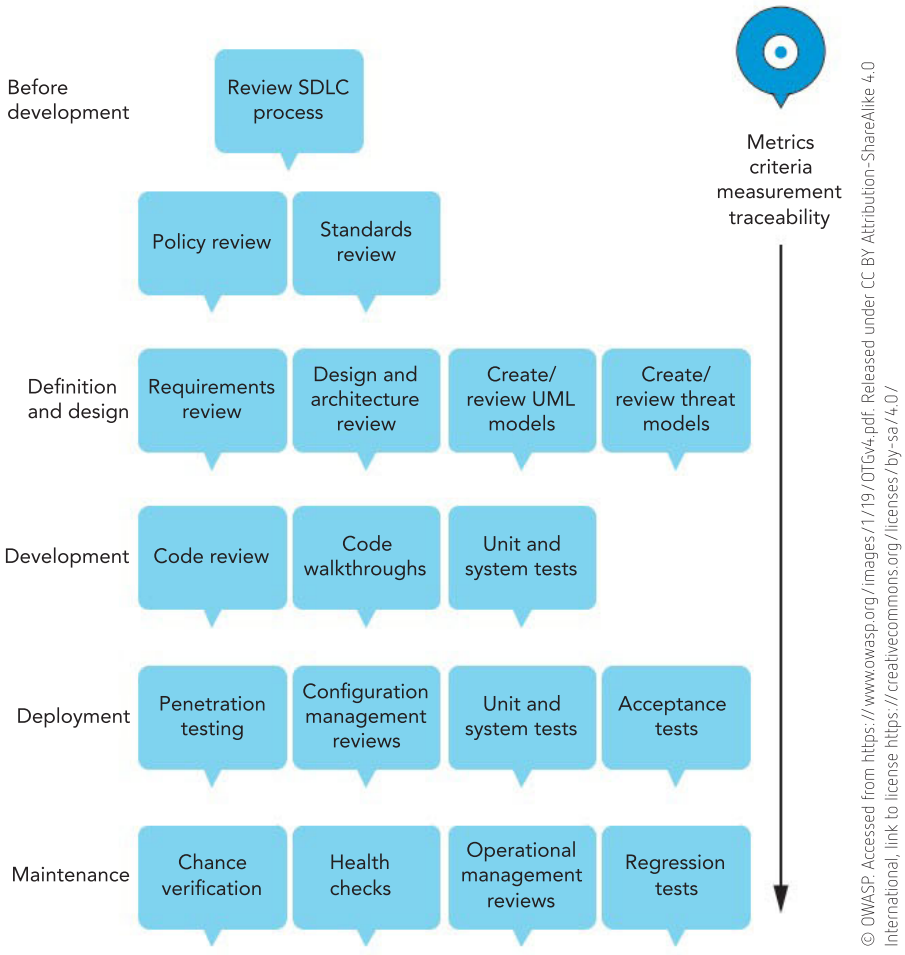
Strategies for minimising potential risks

A recommended attitude to security would be that risks can be reduced rather than eliminated. When new threats and variations on existing threats are revealed, quite often the basics need to be checked, and well-known, established solutions implemented.

Software auditing and testing strategies

During software development there are many opportunities to test the software for compliance with expected outputs. An end-to-end strategy ensures that testing takes place at every stage of the software development cycle. The **Open Web Application Security Project (OWASP)** has published a recommended testing framework that may be considered to form the basis of the development workflow. OWASP provides software tools that can perform tests for known vulnerabilities. Sample code is provided which will 'break' the web application.

OWASP



© OWASP. Accessed from https://www.owasp.org/images/1/19/OTGv4.pdf. Released under CC BY Attribution-ShareAlike 4.0 International, link to license https://creativecommons.org/licenses/by-sa/4.0/

FIGURE 6.17 The OWASP Testing Guide outlines a comprehensive security strategy for the software development life cycle.

SCHOOL-ASSESSED TASK TRACKER

Project plan	Justification	Analysis	Folio of alternative designs ideas	Usability tests	Evaluation and assessment	Final submission

9780170440943

A cautionary comment by Michael Howard, a software security expert at Microsoft, at an OWASP AppSec Conference held in Seattle (2006) was:

> 'Tools do not make software secure! They help scale the process and help enforce policy.'

The Foreword to the OWASP Testing Guide also warns that companies should not rely on OWASP's security software alone:

> Most importantly, these tools are generic - meaning that they are not designed for your custom code, but for applications in general. That means that while they can find some generic problems, they do not have enough knowledge of your application to allow them to detect most flaws. In my experience, the most serious security issues are the ones that are not generic, but deeply intertwined in your business logic and custom application design.

Quote from Michael Howard at the 2006 OWASP AppSec Conference in Seattle. Accessed from https://www.owasp.org/index.php/Testing_Guide_Foreword.
Released by CC BY-SA 4.0 International, link to license: https://creativecommons.org/licenses/by-sa/4.0/

Software auditing

A software security audit may be conducted separately or as part of a larger overall software audit. Auditing of software may have several meanings. One meaning looks at how the software works. Another meaning is to review the software for compliance with specified standards. Does the construction of the software follow the rules? Has any licencing been noted and recorded? For example, the costs of licensing should be looked at when a site licence is held for 10 seats but only six people are ever using the software, or when a licence for 10 is being paid for but 30 people are using the software every day. This issue is not about how the software works; rather, it is a consideration of the risk of legal action and cost benefit of a particular software solution.

The purpose of a software review is to:

- uncover any issues or problems early; it is simpler and costs less to fix an issue earlier in the project
- improve performance, scalability and reliability
- review any necessary or unnecessary testing
- ensure the application can be maintained and extended in the future
- make sure you use the appropriate technology for the job
- satisfy legal and licensing requirements.

The software audit may be an internal or external review; that is, the review may be carried out by someone independent of the developer team, or by the developer team itself. The software audit may use analysis tools to gather data on the performance of the software for security or for functionality.

A typical software audit process will usually:

- document all uses and demands of the software
- test the software for standards compliance, also known as pen testing
- identify assets necessary for the operation of the software, including extreme case limits
- identify security configurations and compare with acceptable security settings
- consider levels of user training necessary for the operation of the software.

9780170440943

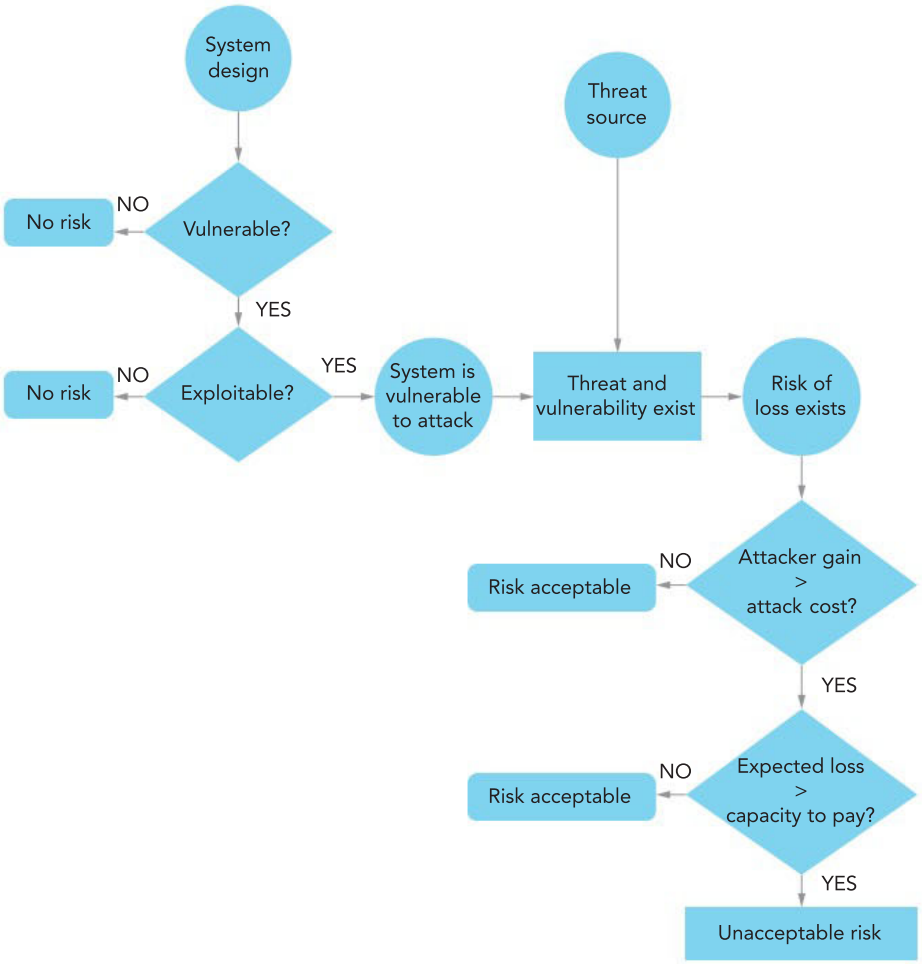


FIGURE 6.18 Determining unacceptable security risk

These findings would be documented and a report prepared. A management group would consider the findings and authorise any further action. Part of the consideration of the report would be to look at risk management and cost–benefit analysis. The directors would consider the legal obligations, and once satisfied that all legal compliance had been met, gauge the level of risk that the organisation would be prepared to accept.

If it is not possible to eliminate all risks, the question then becomes, 'What level of risk is tolerable?'

Risk tolerance, also known as **risk appetite**, is determined by balancing the expense in terms of financial resources and usability of information assets against financial liability, loss of information assets and reputational damage if the risk is exploited.

The steps in determining unacceptable risk are as follows.

- When a flaw or weakness exists, reduce the chances of the vulnerability being exploited by implementing security controls.
- When the vulnerability can be exploited, prevent the opportunity for attack by applying changes in design or administrative controls, or increase protections.
- When an attacker's gain is greater than the costs of attack, apply measures to increase the attacker's costs or reduce the attacker's gains so the gain is not worth the effort.

9780170440943

- When the predicted losses are beyond the ability to absorb the costs, use technical and non-technical protection to limit the impact of an attack and the amount of losses. For example: at the first sign of a DDoS attack, switch the IP address of the server to maintain connections.

Penetration testing (also known as a pen test)

Penetration testing identifies security vulnerabilities in web applications. This is achieved by challenging every page and line of code in the application for known weaknesses. This could be a very time-consuming process if attempted manually. Fortunately, there are automated security tools that will perform continuous web application penetration tests. These automated tests rely on up-to-date descriptions of the parameters that can cause a vulnerability. Each application must be scanned, or crawled, and each parameter must be tested for compliance with OWASP security standards.

The OWASP guide states that 'the most important thing to remember when performing security testing is to continuously re-prioritize. There are an infinite number of possible ways that an application could fail, and organisations always have limited testing time and resources. Be sure time and resources are spent wisely. Try to focus on the security holes that are a real risk to your business.'

Extract from OWASP 'Testing Guide Foreword', accessed from https://www.owasp.org/index.php/Testing_Guide_Foreword. Released by CC BY-SA 4.0 International, link to license: https://creativecommons.org/licenses/by-sa/4.0/

Identifying software and data vulnerabilities

The best defence is to maintain awareness of the possibilities as they may apply to your immediate and future situation. Many of the vulnerabilities are adapted from analogue frauds, scams and confidence tricks that have existed for more than a century. Deception and misrepresentation is a common factor in a large number of the schemes that have been successful in gaining unauthorised access and performing illegal or corrupting actions with data and information.

The largest single common factor is loss of trust with employees or insiders. More than 40 per cent of all criminal activities are perpetrated by people who have inside knowledge or access to the IT systems.

Data breaches

The number of users of the Internet has increased such that:

- commercial company activities are now online
- organisations use the cloud for storage
- big data is collected routinely across the Internet and social media
- greater amounts of information can be created from data due to improved processing speeds and increased storage
- routine retail transactions of consumer banking and purchasing are carried out online.

Accompanying these increases are announcements of data breaches that have become commonplace. This is partly due to new regulations that require anyone affected to be notified (see Think About Software Development 6.6), and partly because there are more

THINK ABOUT SOFTWARE DEVELOPMENT 6.6

The Australian Notifiable Data Breaches (NDB) scheme was applied from 22 February 2018.

The *Privacy Amendment (Notifiable Breaches) Act 2017* applies to all agencies and organisations regulated under the *Australian Privacy Act 1988*.

'The NDB scheme includes an obligation to notify individuals whose personal information is involved in a data breach that is likely to result in serious harm. The notification must include recommendations about the steps individuals should take in response to the breach. The Australian Information Commissioner (Commissioner) must also be notified of eligible data breaches.'

© Office of the Australian Information Commissioner — www.oaic.gov.au. Extract from 'Notifiable Data Breaches scheme'. Released under CC BY 3.0 AU, link to license: https://creativecommons.org/licenses/by/3.0/au/deed.en

Examples of a data breach include the following incidents:

- a device containing customers' personal information is lost or stolen
- a database containing personal information is hacked
- personal information is mistakenly provided to the wrong person.

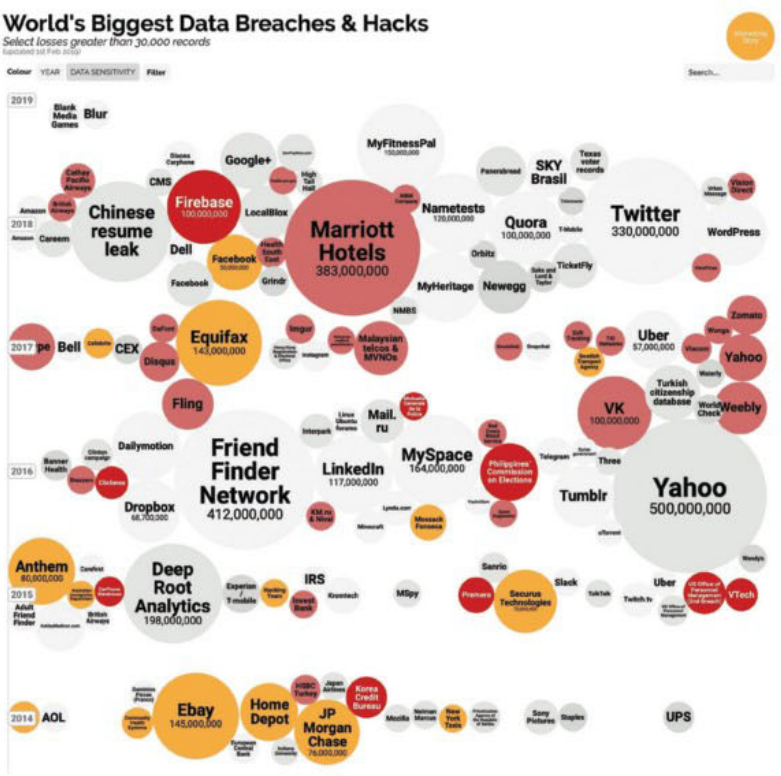
Research the latest (known) data breach locally and globally.

a Identify the nature of the breach.

b How many people were affected?

c What impact will the breach have?

9780170440943



David McCandless via Information is Beautiful

FIGURE 6.19 A selection of data breaches; only those with over 30 000 records are shown.

Breach categories can be filtered on the interactive infographic in Figure 6.19.

breaches being discovered. There are many examples of huge numbers of usernames and passwords being accessed by unauthorised actors. This may come about because:

- a web page looks like a legitimate site and tricks a user into entering their credentials
- there is an automated brute-force attack against a service that does not prevent such an attack
- a service is accessed by an outside party and credentials are stolen, which are then used to access social media and email
- credentials are stolen by malware.

Man-in-the-middle attacks

Man-in-the-middle (MITM) attacks are a type of eavesdropping attack that is difficult to detect. The threat is present whenever a transmission takes place and communications and data are exposed to an unauthorised third party. While unsecured wi-fi is a commonplace opportunity for MITM to occur, many other situations also permit unauthorised interception of the transmission.

In its simplest form, an instance of a MITM attack requires a transmission to be redirected through a third computer system, which the attacker controls, then relayed to either side of a conversation. Often data and information is harvested for valuable or sensitive items, or modified or substituted; for example, bank account login details, company secrets or other confidential material.

SCHOOL-ASSESSED TASK TRACKER

☑ Project plan	☑ Justification	☑ Analysis	☑ Folio of alternative designs ideas	☑ Usability tests	☑ Evaluation and assessment	☑ Final submission

9780170440943

Ways to avoid man-in-the-middle attacks usually involve thorough user authentication. Let's revise the advice from the ACSC to prevent unauthorised access to accounts.

1. Require all users to periodically reset passwords to reduce the ongoing risk of credential compromises.
2. Consider increasing password length and complexity requirements to mitigate the risk of brute-force attacks being successful.
3. Implement a lockout for multiple failed login attempts.
4. If credentials have been compromised, reset passwords as soon as possible.
5. Discourage users from reusing the same password across critical services such as banking and social media sites, or sharing passwords for a critical service with a non-critical service.
6. Recommend the use of passphrases that are not based on simple dictionary words or a combination of personal information. This reduces the risk of password guessing and simple brute-forcing.
7. Advise users to ensure new passwords do not follow a recognisable pattern. This reduces the risk of intelligent brute-forcing based on previously stolen credentials.
8. Use multi-factor authentication for all remote access to business systems and for all users when they perform a privileged action or access an important (sensitive/high-availability) data repository.
9. Look out for unusual account activity or suspicious logins. This may help detect when a service such as email has been compromised and needs a password reset.
10. Encourage users to think carefully before entering credentials:
    - Ask if this is normal.
    - Don't enter credentials into a form loaded from a link sent in email, chat or other means open to receiving communications from an unknown party.
    - Even if the page looks like the service being reset, think twice.
    - Do not click the link. Instead, browse to the website and reset the password from there.
    - Be aware that friends' or other contacts' accounts could be compromised and controlled by a third party to also send a link.
11. If some credentials have been compromised, try to identify a specific cause. Were the credentials entered in an untrusted place? Were they recently reset? What were the credentials for? Were the credentials used elsewhere?
12. Keep operating systems, browsers and plugins up-to-date with patches and fixes.
13. Enable anti-virus protections to help guard against malware that steals credentials.

© Commonwealth of Australia 2019. Australian Government: Office of the Australian Information Commissioner.
Released under CC BY 3.0 AU, link to license: https://creativecommons.org/licenses/by/3.0/au/legalcode

9780170440943

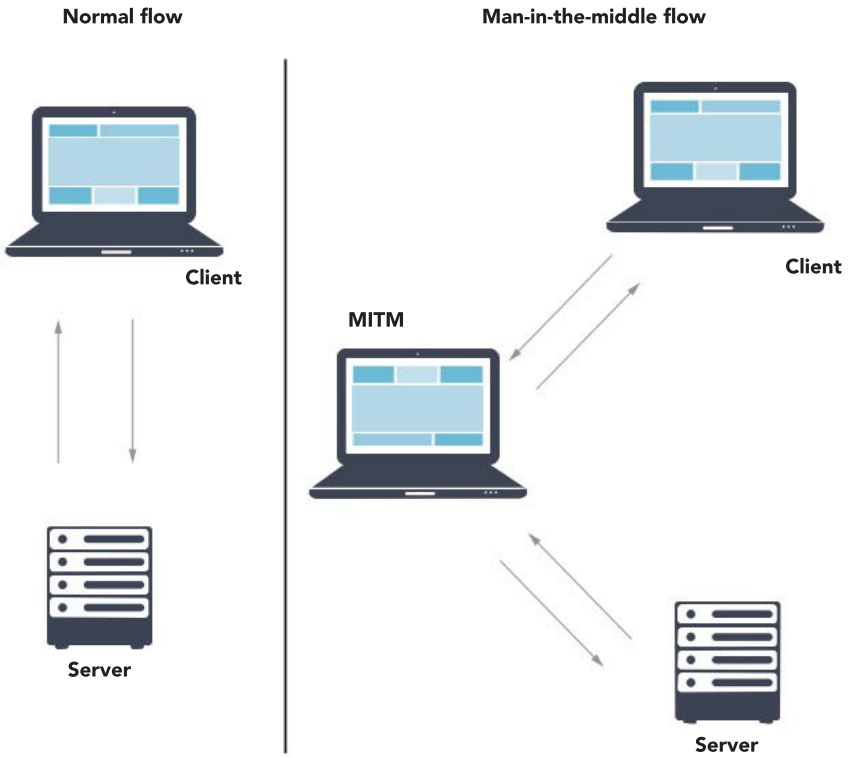


FIGURE 6.20 Man-in-the-middle attacks are very difficult to detect.

THINK ABOUT SOFTWARE DEVELOPMENT 6.7

Describe the steps of a simple MITM attack during an online banking transaction.

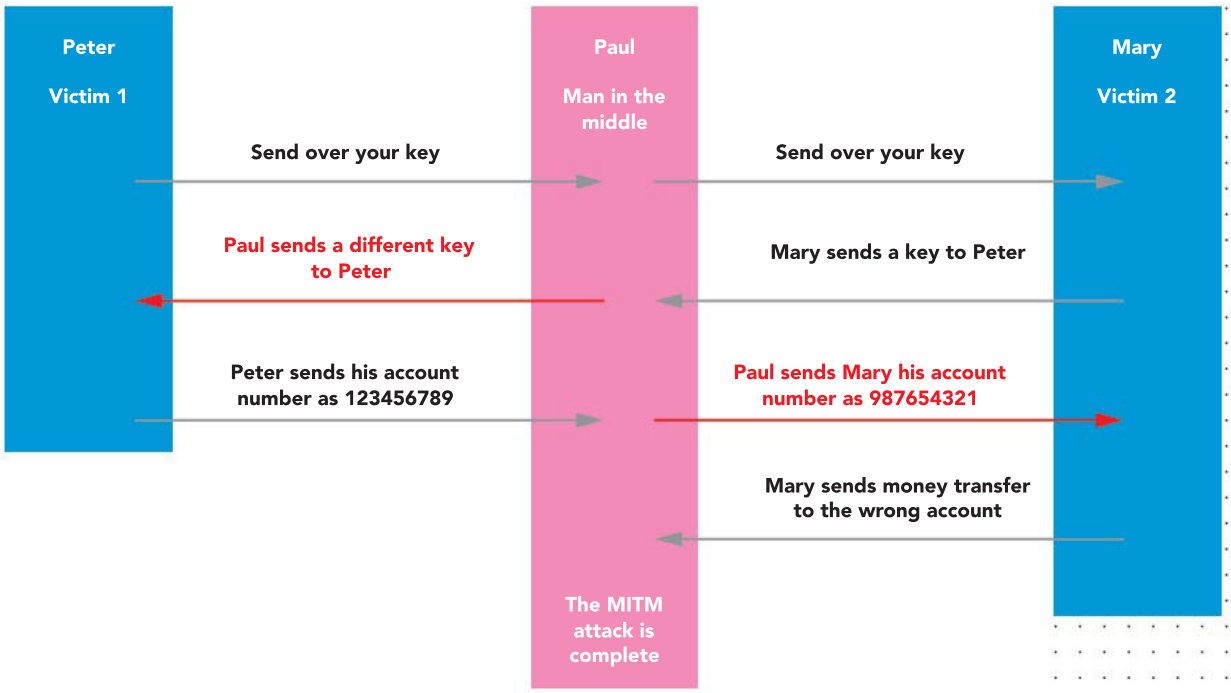


FIGURE 6.21 A banking scam using MITM. This occurs on unsecured wi-fi. Often free wi-fi in shopping centres and airports can be impersonated (faked) and users fooled (spoofed) into using the attacker's wi-fi server.

SCHOOL-ASSESSED TASK TRACKER

Project plan	Justification	Analysis	Folio of alternative designs ideas	Usability tests	Evaluation and assessment	Final submission

9780170440943

Social engineering

In many ways, social engineering attacks are not about computing. **Social engineering attacks** rely on the manipulation of human nature to persuade the victim to provide personal information or to initiate a transaction. Once the attacker convinces the victim that they can be trusted, they gain the management and control needed to achieve the illegal activity.

The basic tricks of the 'con' artist are employed, using flattery and friendliness to gain trust. Another tactic is to create the illusion that 'everyone is doing it' in order to encourage the victim to join in.

Types of social engineering attacks include:

- **Phishing** steals personal information such as usernames, passwords, phone numbers, tax file number, driver's licence number, credit card and banking details. The phishing methods deceive victims into providing personal details by using fake websites or urgent messages requiring a response, or malware that searches the computer and extracts documents and files from the documents directory.
- Spear phishing targets wealthy individuals or victims with valuable access or information.
- Pretexting fabricates a plausible scenario that fools a victim into revealing sensitive information or providing access to restricted systems.
- Baiting entices users to provide their password in exchange for free music or movie downloads. This frequently takes the form of a USB stick that installs malware or a keylogger once connected to a computer or network.
- **Pharming** redirects users to false websites that imitate the legitimate URL. Pharming can affect large numbers of users simultaneously by a 'poisoned' DNS server that re-directs to the wrong website.
- Quid pro quo attacks offer a benefit in exchange for information. These are often in the form of IT service assistance that requires the installation of a 'fix', which is actually malware.
- Tailgating or piggybacking allows unauthorised people into restricted areas. This may take place at security doors or by observing password entry at login screens.

Is there protection against social engineering attacks?

Users must take every online request seriously and treat each one as a potential threat from someone trying to compromise their data. Most social engineering attacks rely on vulnerabilities in human behaviour and poor security habits. In an attempt to overcome the limitations of humans, certain procedures can be adopted.

- Train all users so there is awareness of the risks and an ability to recognise an attack, and the indications of an attack.
- Make sure users know how to report an incident and who to report it to.
- Keep anti-virus software up to date, and activated.
- Use **spyware** detection software.
- Maintain operating systems and applications with updates and patches, after receiving the advice of system administrators.
- Do not open email (or SMS) attachments when the source is unknown.
- Avoid social engineering attacks by refusing to disclose information by phone or email.
- Only download, and install, authorised software from approved download sites.
- Use complex two-factor authentication and protect passwords and usernames.

9780170440943

Strategies to protect against web application risks

Why are web applications vulnerable? The basis of HTTP is clear text. Attackers therefore find it very simple to modify the parameters and execute functionality that was not intended to be a function of the application.

There are known vulnerabilities that simple programming practices can reduce. However, it is surprising to discover that many exploits are variations on known vulnerabilities that were identified many years ago. Many newly published websites just do not apply best practice, or the developers may not have heard of those vulnerabilities. Consequently, their organisations have not been provided with the necessary information and guidance for awareness.

Programming languages used for web applications have many issues in common. A reasonable prediction would suggest that the languages most at risk are the older, more established programs with widespread adoption across many industries and users. The most frequent reported vulnerabilities reflect this prediction. The programming language C was developed in 1973 and quickly spread during the 1980s to provide the basis for solutions across every area of business and consumer computing. PHP was released in 1995 and quickly became the basis for the majority of online commercial websites. In 2019, C and PHP have the highest reported instances of intrusion.

In summary, no user input can be trusted. This means always validating input for type, length, format and range. The data is challenged each time it is processed. For example, when data is entered through a web form to an application script, it will be encoded before being displayed on a dynamic page. Logistically, this means that any values accepted from the client side are checked, filtered and encoded before being passed back to the user. Any user-supplied data handling and processing must be certified as secure.

Web application risks

Assessing web application risks is essential, as an application on the web is exposed to millions of users through the Internet. The first step is to identify the threat, then the counter measure to prevent the exploit being activated. Several threats are well known and well documented.

Cross-site scripting (XSS)

Cross-site scripting (XSS) is the most prevalent web application security flaw. The popularity and frequency of the XSS attack is due to the widespread opportunities provided by so many websites that do not apply basic security procedures. XSS has been well known for many years, but new websites appear every month with dated vulnerable code.

Dynamic and interactive websites are frequently exposed to the threat of cross-site scripting. XSS flaws occur when an application includes user-supplied data in a page sent to the browser without properly validating or escaping that content prior to acceptance at the server. The use of XSS is relatively simple, as it allows malicious active script to be inserted into a regular web page form and instructions issued to the server to provide access or information.

XSS attacks can:

- steal data
- hijack a session and take control of the account

9780170440943

- install and run malicious code
- be part of a phishing scam.

XSS attacks usually take the form of embedded JavaScript; however, any active script is potentially compromising. This includes C++, PHP, Python, Ruby, ActiveX, Visual Basic Script (VBScript) and Flash.

There are three known types of XSS flaws:

1. *Stored attacks* are those where the injected script is permanently stored on the target servers, such as in a database, message forum, visitor log or comment field. The victim then retrieves the malicious script from the server when it requests the stored information. Stored XSS is also sometimes referred to as Persistent or Type-I XSS.
2. *Reflected attacks* are those where the injected script is reflected off the web server, such as in an error message, search result, or any other response that includes some or all of the input sent to the server as part of the request. When a user is tricked into clicking on a link, submitting a specially crafted form, or even just browsing to a malicious site, the injected code travels to the vulnerable website, which reflects the attack back to the user's browser. The browser then executes the code because it came from a 'trusted' server. Reflected XSS is also sometimes referred to as Non-Persistent or Type-II XSS.
3. *DOM-based XSS* or client-side XSS is where the local user browser is modified so that it behaves in an unexpected manner. The appearance of the page does not change, but the page executes differently due to modifications that have occurred in the client-side browser environment. This type of attack is different other XSS attacks, which are due to a server-side flaw. This attack is sometimes referred to as a Type-0 XSS.

SQL injection

Structured query language (SQL) is a standard set of instructions that can manipulate a database server. A very large number of websites have access to database repositories of information and file collections. The usual procedure for requesting information from a database is to enter characters into a form, which is then passed to the database for action.

Control characters have been permitted to be entered and accepted. The preventative measures are relatively simple, ensuring all user input is restricted to acceptable characters, with no delimiting control characters.

For example:

- SQL injections take advantage of special coding characters to break out of data access and switch into code access. The result of injection could be data stolen, modified, corrupted or deleted, denial of access or loss of accountability. Injection can be prevented by keeping untrusted data separate from commands and queries. A secure interface is the most common preventative measure.
- SQL injections have language-specific requirements. JAVA, .NET, PHP and SQLite have well established libraries of suitable preventative procedures.

This is sometimes described as a 'tautology attack', where the condition is always true. This code injection masquerades as a query, but it will always request all records, or usernames and passwords, or admin control of the database.

To avoid SQL injection flaws, developers must i) not use dynamic queries and ii) prevent user supplied inputs, containing malicious SQL code, from affecting the logic of the query.

9780170440943

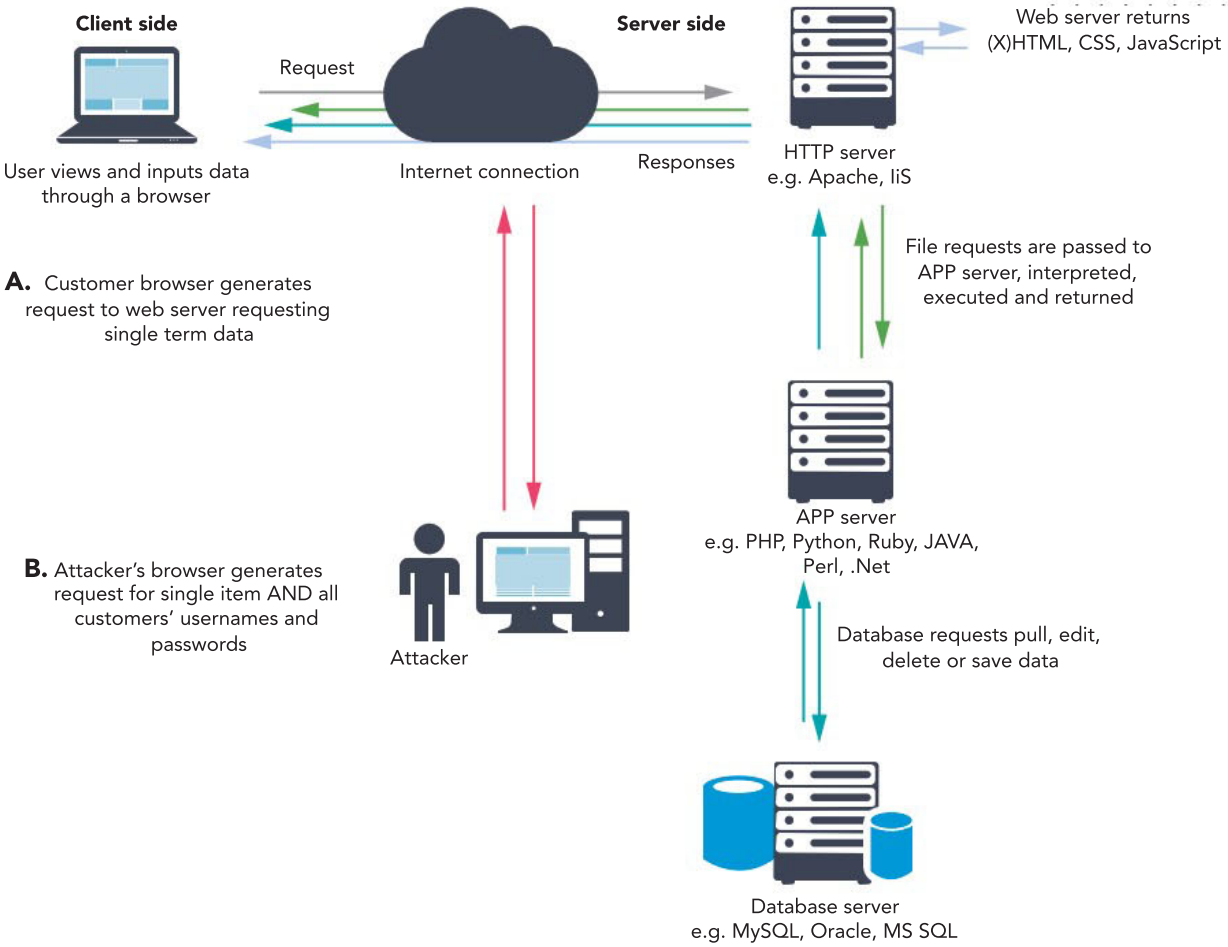


FIGURE 6.22 A standard request for records from a database

The application uses untrusted data in the construction of the following vulnerable SQL call:

```
String query = "SELECT * FROM accounts WHERE custID='" +
request.getParameter("id") +"'";
```

The attacker modifies the 'id' parameter in their browser to send: ' or '1'='1. This changes the meaning of the query to return all the records from the accounts database.

```
http://example.com/app/accountView?id=' or '1'='1
```

In the worst case, the attacker uses this weakness to modify or delete data or perhaps invoke special stored procedures, which could allow a complete takeover of the database host.

FIGURE 6.23 An example attack scenario

SCHOOL-ASSESSED TASK TRACKER

☑ Project plan	☑ Justification	☑ Analysis	☑ Folio of alternative designs ideas	☑ Usability tests	☑ Evaluation and assessment	☑ Final submission

9780170440943

Appropriate measures to prevent SQL injections are to:

- validate all user input before passing the query to the SQL database
- use only prepared statements for database queries to eliminate unvalidated user input. Parameters can be adjusted for narrowing any query
- use stored procedures for users to choose from, rather than entering untested inputs
- whitelist input validation, which requires input redesign to ensure query names come from established names rather than input by the user.

Insecure cryptographic storage

Attackers gain access to sensitive financial or personal subscriber details when data is stored inappropriately or illegally. For example, in Australia, credit card details are not to be kept after a transaction is completed.

During transmission on a secure connection (HTTPS), the data is encrypted and not sent 'in the clear'. Details are encrypted or hashed before storage, so that if security is breached the thieves cannot make sense of the data.

Cross-site Request Forgery (CSRF)

CSRF occurs when a user's browser is spoofed into logging into a site with an unauthorised user's credentials.

Distributed denial of service attacks (DDoS)

A **distributed denial-of-service attack (DDoS)**, or the lesser version, a denial-of-service attack (DoS), involves an overwhelming number of access requests being put to a web server. The attacker sometimes makes use of **botnet** zombies, which have been commandeered unknowingly and are marshalled in a coordinated attack against a specified target. Zombies in a botnet are instructed to send IP requests to a targeted web server simultaneously. The server becomes overwhelmed by the huge number of access requests, and this slows all server operations or causes the server to stop altogether, or crash.

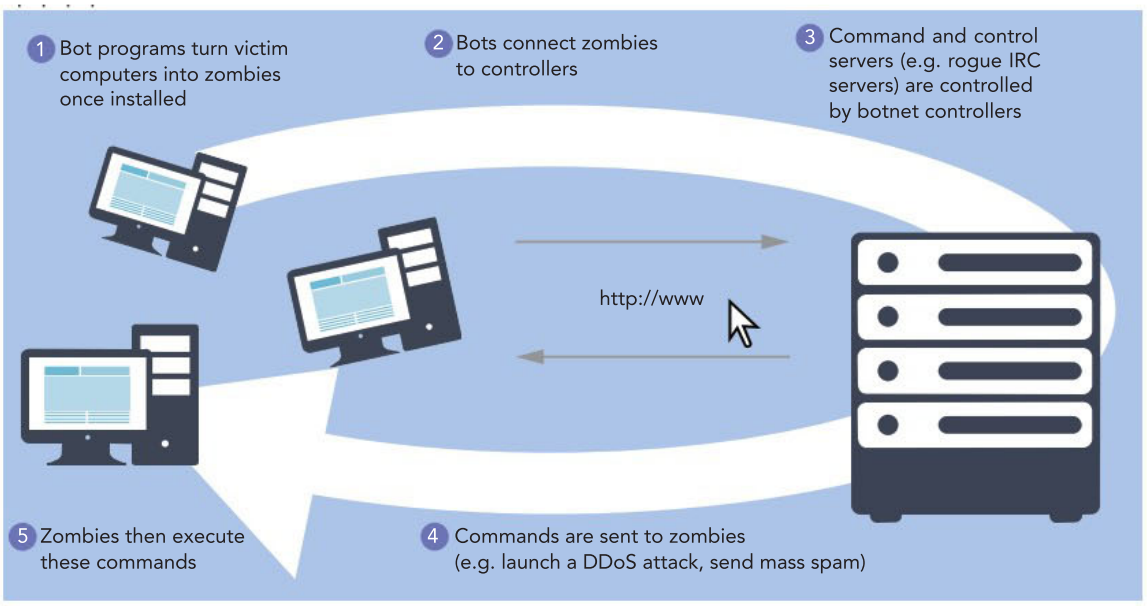


FIGURE 6.24 Denial-of-service attacks are increasingly from cloud-based data centres.

DDoS attacks do not attempt to steal or corrupt data; the purpose is simply to overload web servers and cause them to be unstable. This loss of service can be very expensive to a company or institution. One possible solution for a single server is to change the IP address. It can take up to 72 hours to propagate a new IP. However, this may not stop the problem, as the attackers can simply redirect the attack on the new IP. Other options are to operate several servers with different IP addresses; to increase bandwidth to accommodate 'spikes' in traffic; and configuring firewall/router settings to reject certain DNS and ping-based attacks by blocking certain ports. Automatic active solutions monitor the number of incomplete connections and cancel them when a predetermined threshold has been reached. This will avoid the server being overwhelmed with connection requests.

XML injection or XPath injection

Extensible markup language (XML) is similar to HTML (hypertext markup language); however, it is designed to carry data rather than instructions for displaying web pages. XML is less structured than HTML, and allows for users to define tags. This provides a criminal opportunity to substitute malicious tags in the data to take control of the web page. The **XPath injection** exploits the XML queries that require user inputs.

Bots and web scraping

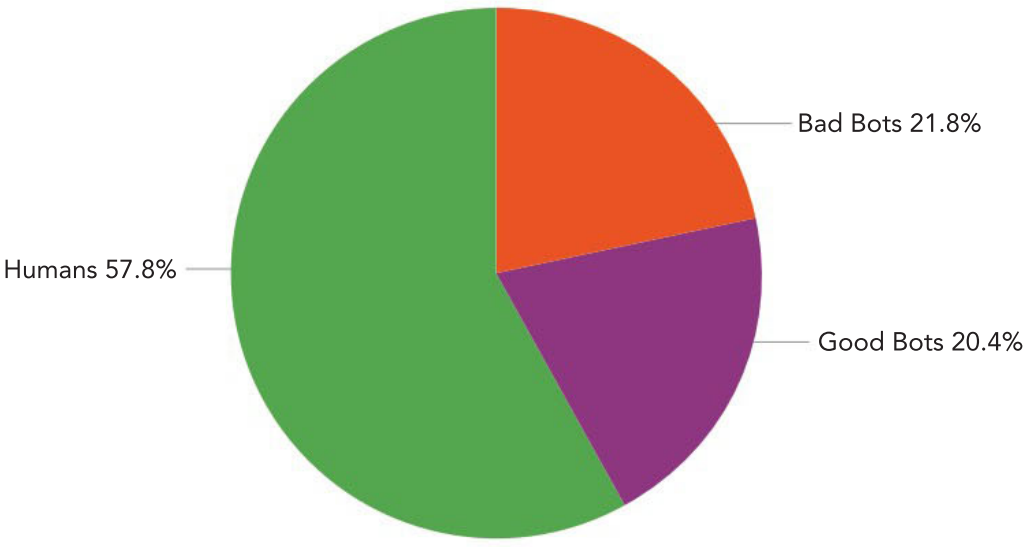
Overall, automated Internet robots, or netbots, are responsible for approximately 40 per cent of all web traffic. While there are good and bad bots, harmful bots are responsible for 29 per cent of web traffic. The bad bots include spambots and unauthorised data scrapers. **Spam** generated automatically accounts for 80 per cent of the email messages circulating each day.

So-called 'good bots' are essential for the orderly conduct of many web functions. For example, Google and Facebook operate many millions of web crawlers or data-extracting spiders that check the connections and update details every day. Feed fetchers are helper bots that update a Facebook feed on a mobile app and cause about five per cent of all web traffic.

There are examples where thousands of bots every day send millions of unauthorised spam emails to make up over half of all Internet traffic.

- 54 billion spam messages per day
- 156 million phishing emails per day

The most common spam email is healthcare or dating spam.



Global Dots 2018 Bad Bot report

FIGURE 6.25 Findings from the Global Dots 2018 Bad Bot report

SCHOOL-ASSESSED TASK TRACKER

Project plan	Justification	Analysis	Folio of alternative designs ideas	Usability tests	Evaluation and assessment	Final submission

9780170440943

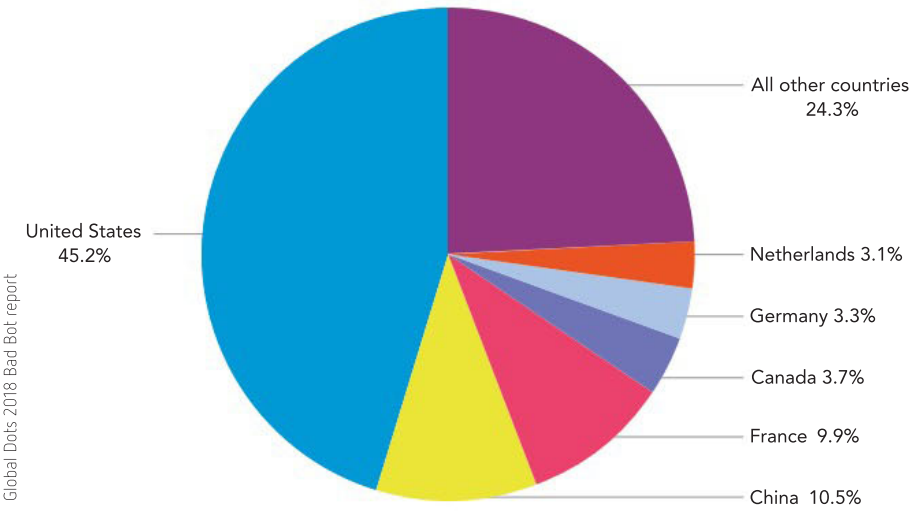


FIGURE 6.26 Source of bot attacks by country. The cyber criminals could be anywhere in the world, but choose to use an IP address that is less likely to be blocked, or is just cheaper.

Impersonator bots, also known as zombies, are controlled by attackers to perpetrate DDoS attacks against third parties. An estimated 25 per cent of web traffic is used in this way. The bot may operate without the knowledge of its owner host. It may be created by malicious code, which creates undetected activities to be carried out by the captured computer. The only way to detect this presence is to monitor computer activity for unexpected online traffic or CPU loads.

Denial of inventory

Bots hold items in shopping carts, which prevent access to items for 'real' customers. The shopping cart is later abandoned as the 'hold' times out. Retailers notice increases in the number of abandoned shopping carts from certain 'zombie' accounts with no purchases.

Gift card balance checking

Bots steal money from gift card accounts with a remaining balance. This is a variation on brute force or dictionary attacks, and can be detected by spikes in login requests to gift card balances, and resultant complaints from customers who have 'lost' the balance of their account.

Price scraping

Web scraping is the act of taking data and information without authority or permission. Often a website display relies on the application of copyright laws to protect ownership access and use of the information and data. Data scraping aggregates enormous volumes of data and applies the knowledge gained for commercial advantage. For example, price scraping of competitors' websites ensures that up-to-date knowledge can be used to undercut and the perpetrator can emerge with the lowest prices in the marketplace. Airlines, gambling odds, real estate, used-car prices, petrol and other supply retailers are all business categories where price scraping occurs every minute of every day.

Content scraping

Content scraping is when web page content is stolen and reused without permission. Although copyright laws apply, the fact that the Internet is international means it is unlikely

THINK ABOUT SOFTWARE DEVELOPMENT 6.8

Research the latest Bad Bot report.

1. Identify the most recent numbers of Humans vs Good bots vs Bad bots.
2. Consider whether any categories of attack have changed.
3. Has the source of bot attacks changed?

that any action will be taken. Another aspect of content scraping is to save on the cost of gathering data and information and take advantage of the infrastructure of another business. Examples are news gathering, textbook and subject reference websites, sporting results websites and stock market 'ticker' websites where summary reports of activity are available for a fee. Paywalls and login requirements offer little defence against aggressive web scraping. The web bots can mimic human behaviour, create login accounts and extract large amounts of content. Previously, displaying the web page as an image was an effective defence against bots that would record text; however, AI and OCR converting text within an image have made this a trivial conversion task.

An example of a web scraper application is when the application can record a specific stock price every five minutes and the volumes traded without the need to subscribe to a stock market monitoring service. Similar monitoring of airline prices for ticketing can be automated. Images, email addresses, phone contacts and addresses can all be harvested from websites automatically.

Strategies to reduce the occurrence of bot attacks

- Identify out-of-date browser agents, which often disguise legacy bots. Access to the website can be denied by blocking superseded versions of common browsers; for example, versions of Firefox, Chrome, Internet Explorer and Safari that are more than two years out of date. An alternative is to insist on human logic for login access, such as using the captcha process.
- Monitor website traffic. Spikes in traffic can be assessed and the sources identified. If a single specific source is indicated, then a bot is responsible for the increase in apparent business. Failed logins may also indicate bad bot activity.
- Be alert for public data breaches. Whenever credit cards details are stolen, they are quickly deployed to run the credentials into your website. Be aware of financial authorities' warnings and updates to ensure credit cards are genuine.

Session hijacking

The integrity of transactions across the web depends on certainty of identity – if users are who they say they are. Usually when an account holder logs in, a session token is issued. All transactions that take place in that login session exchange the token to verify the identity of the user. Session hacking is a variation on eavesdropping, where the attacker attempts to impersonate the legitimate user. Benefits of session hijacking depend on what is dealt with in the session, which typically ranges from shopping carts to credit card details.

The vulnerability is caused when session identifier tokens are stored or transmitted insecurely. A simple solution is to ensure that the Internet connection is secure and uses the encrypted protocols HTTPS and SSL.

Buffer overflow

When a process attempts to store data in RAM beyond the assigned fixed length storage buffer, an overflow into neighbouring memory locations may occur. Under certain conditions the overflow data may corrupt data, or it can stop functioning by crashing the computer or being replaced with attacker code to provide alternate instructions for execution once the program restarts.

9780170440943

Buffer overflow attacks are well known, but legacy and newly developed applications continue to be vulnerable. The basis of the success is violation of a programmer's assumptions.

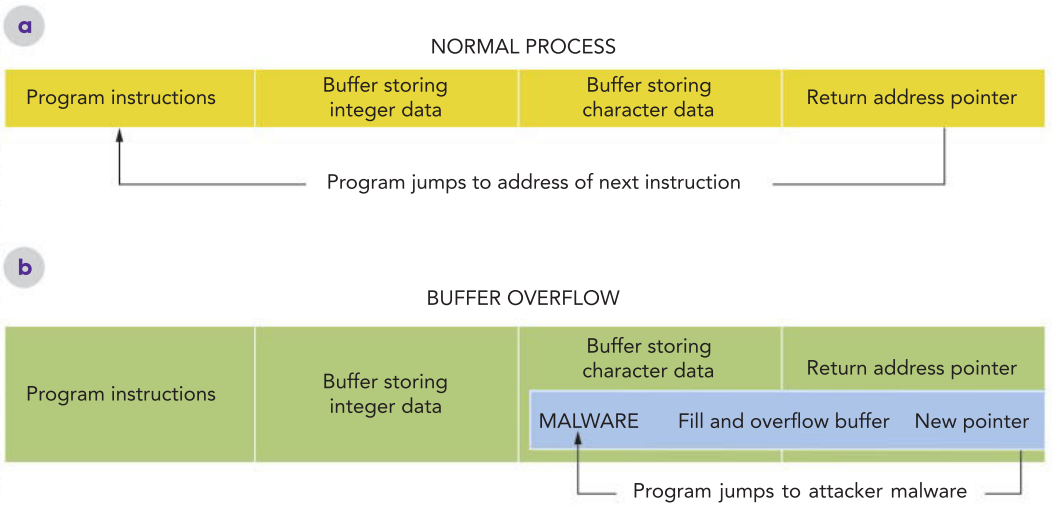


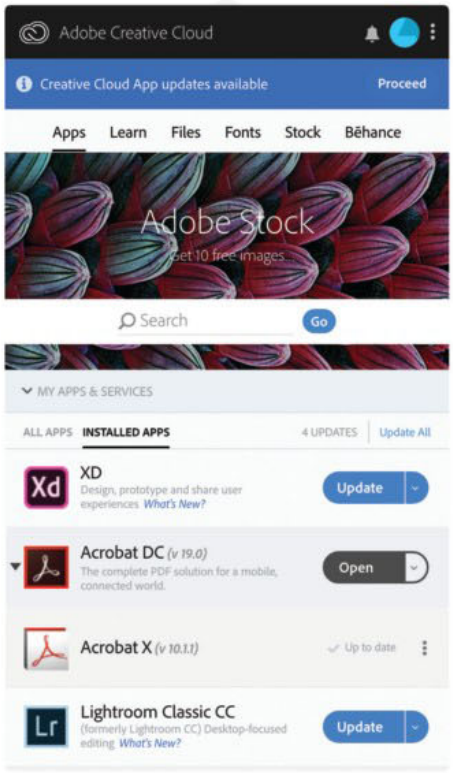
FIGURE 6.27 **a** The normal process; **b** buffer overflow. The simplest buffer overflow attack is the most common.

Protection against buffer overflow attacks can be provided by ensuring all code that accepts input from users via an HTTP request is reviewed to make sure it provides appropriate size checking on all such user inputs.

Java and Python are interpreted programming languages, and are immune to these attacks.

Restrict URL access

Undertake access control checks each time a page is accessed. This will prevent attackers entering URLs directly into browsers, meaning hidden or orphan pages cannot be accessed, as permission will be denied to unauthorised requests.



Adobe product screenshot reprinted with permission from Adobe.

FIGURE 6.28 Adobe Creative Cloud is a subscription service.

Software acquired from third parties

Many software products are offered as a subscription rather than as a 'purchase'. Traditionally, a software purchase was a payment for a licence to use the software on a 'as is' basis. The software was not permitted to be modified or re-engineered.

One direct result of the subscription model is to maintain the distributed user base of software and ensure software is up to date. Automatic software updates prompt the user to ensure that the latest security patches will be installed.

Other software purchased and installed may have been registered with the developer. This registration usually provides update notifications or access to a web page that provides information about security issues and fixes.

9780170440943

Overall, the user is reliant on the software developer or distributor for notification about necessary improvements and modifications to the original product.

Web applications often rely on third-party services and applications. For example, Google forms is often used to host a survey activity which is used for purposes unrelated to Google. Disclaimers warn users that their details are not secure.

From Google Terms of service: Software ... You may not copy, modify, distribute, sell, or lease any part of our Services or included software, nor may you reverse engineer or attempt to extract the source code of that software, unless laws prohibit those restrictions or you have our written permission ...

Extract from Google terms of service: Software, accessed from https://policies.google.com/terms?hl=en

© OWASP. Accessed from https://www.owasp.org/images/1/19/OTGv4.pdf. Released under CC BY Attribution-ShareAlike 4.0 International, link to license https://creativecommons.org/licenses/by-sa/4.0/

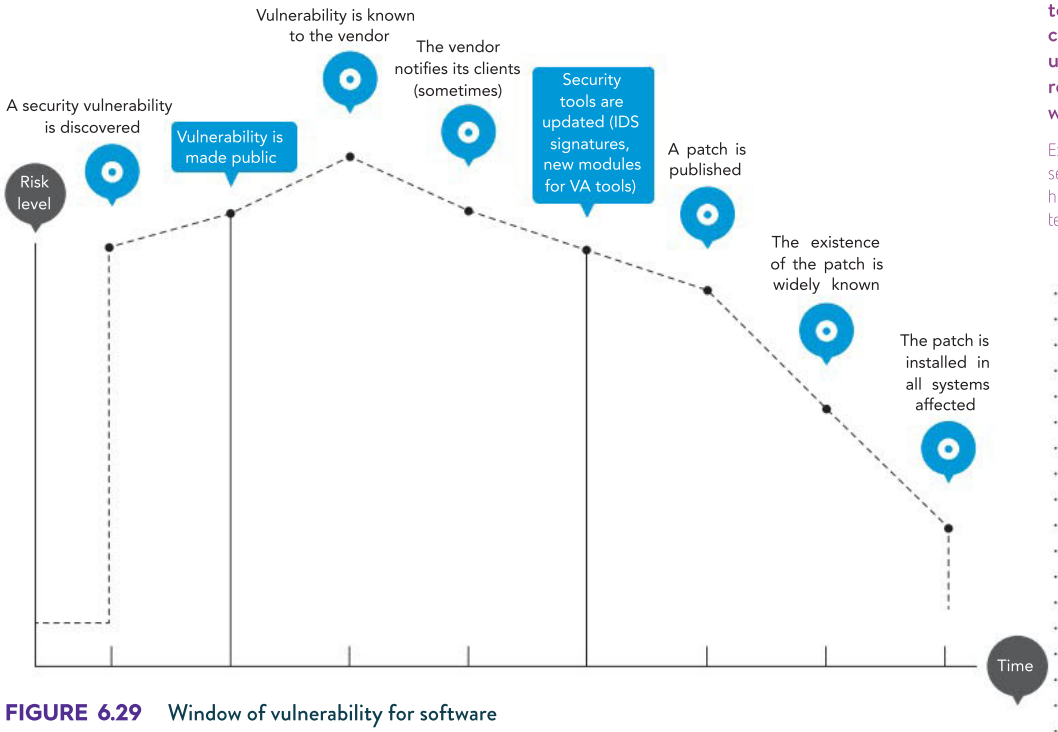


FIGURE 6.29 Window of vulnerability for software

Integrity of data

A secure system must ensure that confidentiality, integrity and availability are maintained. This CIA data security systems approach was previously described on page 215 and shown in Figure 6.2. A further consideration on the security of the system is the data held within and generated by the system. Every interaction with a computer system requires data certainty. Without an unblemished level of trust, doubt and uncertainty will develop to the point where that system, consisting of software and hardware components, must be abandoned, banned or blocked to avoid further disruptions, losses and contamination. For example, many websites are blocked by ISP and networking administrators (i.e. 'blacklisted') because they are a known source of malware including trojans, worms and/or adware.

Threats to data integrity

Threats to data integrity can be accidental, event-based or deliberate.

Accidental threats

There are several ways users can cause accidental damage to information systems. It is important to make a distinction between inadvertent errors, which may be unlikely or

SCHOOL-ASSESSED TASK TRACKER

☑ Project plan	☑ Justification	☑ Analysis	☑ Folio of alternative designs ideas	☑ Usability tests	☑ Evaluation and assessment	☑ Final submission

9780170440943

unforeseen, and those errors that occur due to poor training or poor design of user interfaces (UI). Mistakes rarely 'just happen'.

Mitigating circumstances or contributing factors often include:

- user inattention or carelessness
- confusing screen design with lookalike interface
- lack of confirmation before execution; all changes are 'live' with no simple backup process
- inappropriate permissions, which allow untrained users access to modify or delete strategic files or settings.

Event-based threats

Event-based threats are usually external to the organisation and beyond the immediate control or influence of security planning to prevent. The only course of action is to prepare contingency plans in the event that the threat occurs. Well-considered recovery plans can minimise the impact on the organisation.

Event-based threats include:

- failure of storage; e.g. HDD or SSD
- power failure
- file corruption
- using cloud storage where power outage, software malfunction or data breach occurs
- using third-party software where software malfunction or data breach occurs
- acts of nature; for example, fire, flood, earthquake, lightning strike/power surge.

Similar strategies can be used to prevent accidental loss or event-based damage.

A **hot site** maintains a full or partial duplicate for a primary IT operation, including complete computer systems and near-real-time backups for systems, applications and data. In its most expensive form, mirroring software is used to keep a hot backup site and a primary site synchronised. A hot site is used when an organisation can tolerate little or no downtime, and is where a switchover may be achieved in a few hours.

A **cold site** by contrast may take a few weeks to be established following a catastrophic systems failure, though the business may be able to operate at a reduced level without full IT services. Increasingly, businesses are more dependent on IT, so **warm sites** (also known as mobile sites) become attractive. The time for startup depends on how quickly access to the restored backup can be made operational.

Deliberate threats

Many of the details of deliberate threats have been discussed earlier in this chapter. Table 6.5 provides a summary of those threats, prevention strategies and actions to be taken. One particular security strategy, known as a **honeypot**, has been widely adopted to detect, deflect or counteract unauthorised access to computer systems. The honeypot acts as a decoy for trapping hackers' or tracking new hacking methods, or simply as a delay by wasting hackers, time. Honeypots are established solely to take attention away from the main company servers. They have no other purpose, and legitimate users will not connect with the honeypot. Attackers can be identified and traced if a honeypot has logged the attacker IP details. Multiple honeypots across a network may form a **honeynet**. Honeypots can also be used on an internal network to prevent authorised users browsing beyond their immediate permissions.

9780170440943

TABLE 6.4 Summary of event-based threat detection, prevention and recovery actions

Detection	Prevention	Prevention action	Recovery
Unexpected computer system behaviour: • data loss • slow performance	Ensure diagnostic monitoring operates continuously, regularly checking condition of storage media. Periodic full backups are enabled. Incremental backups with short intervals are enabled, dependent on activity and cost effectiveness.	Maintain RAID and incremental backup protocols. Diagnostic tools constantly monitor hardware and software. Alerts are issued when anomalous events occur. Uninterruptable power supply (UPS) provides sufficient time to save all data and settings before controlled shutdown.	Restore data with most recent backups. Replace damaged, lost or stolen equipment. Activate disaster recovery plan (DRP) to limit losses and damage.
Disaster alert from emergency services, smoke alarm	Ensure regular backups are recorded. Establish off-site duplicate of IT services when the base operation is unavailable.	Fire services deployed appropriate to the circumstances. Fire detection/smoke alarms. Halon gas reduces potential for flame and water damage. Uninterruptable power supply (UPS) provides sufficient time to save all data and settings before controlled shutdown.	Consider hot or cold site recovery plan solution. Restore data with most recent backups.

TABLE 6.5 Summary of deliberate threat detection, prevention and recovery actions

Detection	Prevention	Prevention action	Recovery
Malicious software or malware: viruses, worms, trojans, logic bombs, spyware, back doors, **keyloggers** may cause: • unexpected file losses or corruption • slowly running computer • unusual file sizes or date/time stamps	Permissions are restricted for all users. Establish download limits and email attachment limits. Ensure all users are trained in recognising the dangers of 'unsafe sources' of software and untrusted sources of documents. Install and run anti-virus software and SPAM filters. Run anti-virus scans on incoming email.	Firewall settings block unauthorised access to the computer or the network. Spam filters are constantly available to scan all email. Set anti-virus software to scan files introduced on USB drives.	Quarantine affected files and any infected computer from the network. Reformat the drives and wipe all data. Restore lost data from latest incremental backup.
Cyber attackers: hackers, spies, insiders, criminals and terrorists; activity is sometimes difficult to detect	Intrusion detection system (IDS) monitors system and file activity to identify and signal an alert whenever anomaly or violations occur.	Constant vigilance is necessary, as detection is slow and false alarms are more frequent than actual attacks.	Data breaches require notification of those affected (see chapter 7). Loss of usernames and passwords will necessitate re-establishing authenticity for users. Suspend all account activity until all passwords are reset through **two-factor recovery** process.
Social engineering attacks: phishing, hoaxes, fraudulent offers, **impersonation**, pharming, **shoulder surfing**, tailgating, spear phishing, vishing, **smishing**; often difficult to detect.	Ensure all users are trained in recognising the characteristics of social engineering and receive regular updates on the most recent versions. Restrict the use of company SMS and email accounts to strictly business purposes. Ensuring user email subscriptions are limited to essential contacts to minimise SPAM.	Social engineering is, at its most basic, about confidence tricks, or the 'con'. Vigilance is necessary at all times with email messages, SMS and websites. Limit financial authority within the organisation to ensure only legitimate orders are taken and payments made.	Depends on the instance, and whether the violation is detected, or reported. Often embarrassment prevents the reporting of these criminal activities.

SCHOOL-ASSESSED TASK TRACKER

☑ Project plan | ☑ Justification | ☑ Analysis | ☑ Folio of alternative designs ideas | ☑ Usability tests | ☑ Evaluation and assessment | ☑ Final submission

9780170440943

Characteristics of data integrity

The factors of data integrity that must be maintained by a properly functioning information system include:

- accuracy
  - correctness
  - completeness
  - consistency
  - clarity
- authenticity
- reasonableness
- relevance
- timeliness.

We shall now consider each of these factors in further detail.

Accuracy

There are two main characteristics of data accuracy: content (functionality) and form (appearance). Content is divided into two parts: correctness and completeness. Form is divided into consistency and clarity. The following sections discuss each aspect of accuracy, followed by challenges to data accuracy and measures to improve them.

Correctness

Correctness means that the values stored for a given object must be correct. For example, if you create a database that includes a list of all the users subscribing to the software that you are developing for Outcome 1, you would need to ensure that you enter each subject's details into the database correctly. The content would be incorrect if you keyed the wrong date of user testing into the TestDate field or misspelled a surname in the LastName field.

Data is also correct if it is a truthful representation of the real-world construct to which it refers. For example, the weather forecasts by the Bureau of Meteorology are considered (usually) correct, in that they truthfully represent our *concept* of the weather in terms of predicted temperatures, rainfall, winds and so on.

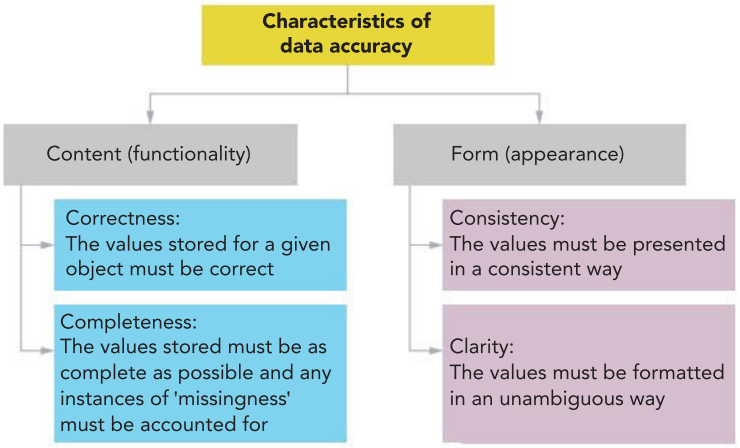


FIGURE 6.30 Characteristics of data accuracy

Completeness

Completeness means that the entire data set is intact – all the data from all programs on all variables at all relevant points in time and space. Completeness can be very difficult to achieve.

Clarity

The form of data is important as well as the content, because this will remove ambiguity about the content. Accordingly, form is divided into two parts: clarity and consistency.

Clarity is about formatting data in an unambiguous manner to prevent misinterpretation. For example, you have users enter dates and they are the correct values. However,

9780170440943

they are still not completely accurate, because some of the dates have been entered using the US date format (MM/DD/YYYY), and others using the Australian date format (DD/MM/YYYY). Worse still, some of the dates have both days *and* months less than 12, making the actual entered dates ambiguous:

```
04/11/2020    Data testing complete
12/02/2018    User test complete
06/09/2019    Data verified
```

The dates are inaccurate because you cannot tell what the true, correct values are. You need to be stricter when entering data. Doing this means you can apply consistency.

Consistency

Correct, unambiguous data can still cause a problem if it is not consistent. Inconsistent data is unwelcome because it means the data is unreliable. This is why consistency is part of data accuracy.

Consistency is also a concern on a larger scale, when it occurs between multiple data sources where conflicting versions of data appear in different places. If a 'true' value cannot be easily determined between multiple data sources, an entire data source loses integrity and becomes tainted.

Authenticity

Data and information are only authentic while the origin attributed to them is correct, and providing that the data has not been improperly or inappropriately changed since it was published.

Digital documents are easy to fake and distort convincingly. It can be difficult to tell if an image, document, database or web page is genuine, a parody (for example, *The Onion* website) or a deliberate attempt to mislead, fool or defraud people, as with spam, links that trick visitors into clicking ads, and phishing sites.

Characteristics of authentic data

Digital data can only be considered authentic (genuine) if it:

- comes from the author and/or source it claims to be from
- has not been deliberately corrupted
- is not faked or disguised as something else
- has not been changed without authorisation
- is what it claims to be and does not misrepresent itself
- does not aim to mislead or deceive by pretending to be anything else
- does not lie.

Reasonableness

It is possible for data to be perfectly valid; however, it may still be unreasonable. Computer systems cannot apply meaning to data unless specifically instructed to examine limitations.

Reasonableness checks or reasonableness tests apply logic to the raw data to consider whether the combinations are plausible or logically possible or satisfy common sense. In order to perform a reasonableness test, an understanding of the extent of the data is first required. Reasonableness checks are usually performed before data processing.

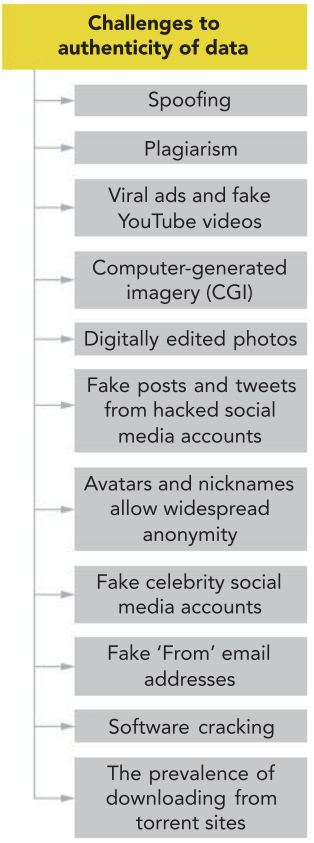


FIGURE 6.31 Challenges to authenticity of data

SCHOOL-ASSESSED TASK TRACKER

☑ Project plan	☑ Justification	☑ Analysis	☑ Folio of alternative designs ideas	☑ Usability tests	☑ Evaluation and assessment	☑ Final submission

9780170440943

Some examples of reasonableness are as follows.

Data is reasonable:

- Medical records have parents and children listed; the age difference is 15–45 years
- Address of student has the same postcode as the school attended
- Tax records have deductions for 500 kilometres per week of work-related car travel
- ABS census indicates there are five persons living at the three-bedroom apartment address
- W-League player has represented Australia 150 times after debuting in 2004

Data is unreasonable:

- Address data and place of work may need to be in the same state, depending on employment
- Medical records have parents and children listed; the age difference is about 10 years
- Tax records have deductions for $1000 per week of work-related internet usage
- ABS census indicates there are 12 persons living at the single-bedroom apartment address
- AFL player has played 450 AFL/VFL games

Relevance

People look for information that relates to a topic that is of interest to them. Relevance measures how closely a resource, such as a document, database or webpage, corresponds to that person's desire for information.

Relevance is not always easy to measure. For example, it *is* obvious that income is relevant to spending habits. However, it is less clear whether age or gender are relevant to development of mental conditions such as schizophrenia. Data about schools in the United States and United Kingdom or Finland may be relevant to Australian schools, but this is not absolute. Data about men may apply to women in some circumstances, but not in others. Assuming that data is relevant when it may not be can lead to invalid conclusions being drawn.

TABLE 6.6 Circumstances when data is in danger of becoming less relevant

Reason	Example
It is on a different topic. Using Internet research to drift from link to link may lead to information that is irrelevant to the original topic.	A discussion of Scottish history mentions the sport of caber tossing, which leads to a discussion of how the sport is scored, which somehow drifts into detailed coverage of kilts.
It is from a place where conditions are not comparable.	Comparing gun ownership in suburban Australia with that of wild Canadian bear country
It is from a different time.	Boys in the 1960s were happy to leave school at 15 and work in the same job for 45 years.
There are significant differences in history, conditions or circumstances that prevent two data sets being compared.	Cultural differences, war status, climate data methods, characteristics of interview subjects and unusual recent events

9780170440943

Timeliness

Data must be timely for it to produce usable information. That means it needs to be processed while it is current and there should be no significant delays in retrieving it. The methods used to process data should be efficient enough to be completed by the time the data is actually needed.

Ensure that the digital systems (hardware and software) used are powerful enough and appropriate for the task at hand to avoid causing delays. It is crucial that decision-making is never based on outdated data.

Users of the data are protected as much as possible from potential delays caused by power outages, hardware failures, downtimes due to system upgrades, and deliberate threats from outsiders, such as denial-of-service attacks or malware infection.

Make sure the age of data is known before using it: it may be meaningless to draw conclusions from very old data; for example, using data about computer use that is 10 years old to draw conclusions about network performance today.

Next steps

This chapter has discussed cybersecurity and some of the main threats to physical and logical (software) security. Software development processes continually prepare for maintenance of confidentiality, integrity and availability. Those entrusted with preparing security plans regularly update their understanding of social engineering, web application threats and the countermeasures required to prevent such attacks. In the next chapter you will consider the legal requirements and obligations for software providers, and those who provide customer services involving personal data and information.

The final assessment for Unit 4 will be Outcome 2, which requires you to consider a case study and provide recommendations on modifications after exploring security and legal implications.

SCHOOL-ASSESSED TASK TRACKER

☑ Project plan	☑ Justification	☑ Analysis	☑ Folio of alternative designs ideas	☑ Usability tests	☑ Evaluation and assessment	☑ Final submission

9780170440943

6 CHAPTER SUMMARY

Essential terms

botnet zombies a coordinated network of compromised or 'zombie' computers used to transmit malware or spam or to launch DDoS attacks

cold site prepared external recovery IT infrastructure; when the duplicate is not immediately ready to replace operations, and may take a few weeks to resume normal operations

crypto jacking / crypto mining hijacking idle processing power of a victim's device and using it to mine cryptocurrency

cybercrime targeted attacks against financial networks, gaining unauthorised access to information and stealing sensitive information

cybercriminals identity thieves, fraud perpetrators, scammers using spam and phishing

distributed denial-of-service attack (DDoS) an attack that overwhelms a web server by targeting requests

exploit take advantage of a vulnerability

hacking unauthorised access to a computer or network or the modification of programs, systems or security for unapproved purposes

honeynet multiple honeypots across a network

honeypot a decoy server that imitates the 'real' website to lure attackers away from production systems to waste time and dissipate their energies harmlessly

hot site prepared external recovery IT infrastructure; where the duplicate is ready immediately to replace operations by restoring backups and resuming normal operations with little or no downtime

impersonation an attack that creates a fictional persona and then plays the role of that person in order to defraud or deceive

keyloggers record keystrokes from a computer and send passwords or sensitive information to unauthorised attackers

malware 'malicious software' designed to infiltrate and damage computer systems without authorisation

man-in-the-middle (MITM) attacks a type of eavesdropping attack where communications and data are exposed to an unauthorised third party

warm site a recovery plan that is in between hot and cold, and limited in scope and scale compared with normal operations; relies on restored backups being brought online before systems and services can be made operational

OWASP Open Web Application Security Program

pharming redirecting users to false websites that imitate the legitimate URL

phishing pretending to be a reputable person or entity in order to induce the disclosure of sensitive information

piggyback entry *see* tailgating

ransomware will lock or encrypt a user's computer until a ransom fee is paid

risk appetite *see* risk tolerance

risk tolerance the quantity and nature of risk that is acceptable

shoulder surfing when an attacker observes password entry or security codes on a keypad, sometimes with a camera

smishing using SMS for phishing attacks

9780170440943

social engineering attacks tricking the victim into clicking 'accept' with admin permissions or into giving the attacker physical access to a device

spam unsolicited bulk messages and advertising

spoofing involves hoaxing, tricking and deceiving users by use of fake usernames or user identities, email addresses or URLs

spyware collects information without the users' knowledge; arrives as a free download and is automatically installed

tailgating when an authorised person swipes their entry card and enters, and an unauthorised person quickly enters behind them before the door closes

trojan like a trojan horse, a program that appears to be safe and reliable but which creates a back door into the computer

two-factor recovery using a recovery email and a phone or SMS message in order to verify or re-establish the identity of the user

version control records each stage of the development so that it is possible to 'roll back' to any previous point in the development

virus often arrives as an email attachment or download, and infects the active host computer and those on any contact list with spam and unwanted ads; hijacks the web browser and disables security access settings

worms standalone self-replicating programs that exploit operating system vulnerabilities

XPath injection an attack technique that is used to compromise the logic of an XML application

zero-day attacks attacks that leave no time to prepare for or defend against the attack

Important facts

1. **Spam** is estimated to comprise up to 80 per cent of the 20 billion messages sent on Twitter each day.
2. **Phishing** attacks are increasing as other methods are defeated by automatic protections; humans are the weakest part of any security system or strategy.
3. Phishing can come in the form of email, SMS, web pages and other social messaging apps.
4. **Spear phishing** is a targeted attack at someone with something of value, which may be financial, data or information about access to other systems.
5. Users should change their passwords regularly, have **unique passwords** for different accounts, take advantage of multi-factor authentication features whenever possible, or use a password manager tool to help securely store credentials.
6. **Social engineering attacks** rely on human weakness, so users must apply critical thinking in their social media consumption, and be diligent in checking whether an email or a phone call is from a trusted source.
7. Smartphones receive **smishing messages** when an SMS message contains an invitation to enter personal information, bank account or credit card details.
8. Users often do not know their computer has been compromised and is acting as a zombie computer. Botnet activity is commonly detected by unusual CPU activity or bandwidth use.
9. **Pharming** can affect large numbers of users simultaneously due to a 'poisoned' DNS server, which re-directs to the wrong website.
10. Computers, tablets and smartphone devices need to be protected from ransomware, dangerous websites, and identity theft threats by using **anti-malware solutions**, including only acquiring apps from trusted sources.
11. **Honeypot** is a decoy server that imitates the 'real' website to lure attackers away from production systems to waste time and dissipate their energies harmlessly. An attacker's IP may be identified and traced. New malware may be captured and identified using honeypots.

9780170440943

TEST YOUR KNOWLEDGE

Review quiz

Physical and software security controls

1 What is a 'zero-day' attack? Why is it so effective for hackers?

2 What is the difference between a vulnerability and an exploit?

3 Only certain users are permitted to view the information because they have:

A confidentiality. **B** availability.
C integrity. **D** authorisation.

4 Defence against attackers is increasingly difficult due to: (more than one answer)

A complexity of attack tools.
B weak patch distribution.
C greater sophistication of attacks.
D delays in patching software products.

5 The process that ensures an individual is who they claim to be is known as:

A authentication. **B** access control.
C certification. **D** verification.

6 Using a brute force attack, what is the average number of combinations that will be attempted in order to crack a cryptosystem that is based on 32-bit key? The estimated number is an average of 1, being the first combination attempted, and the maximum number of combinations, being the last combination if none others were the key. The actual combination will be between the first and last options.

Software acquired from third parties

7 After an attacker has explored a network for information, the next step is to:

A move on to another system.
B modify security settings.
C corrupt networks and devices.
D overcome any defences.

8 Which of the following are not considered 'insiders'?

A Business partners **B** Sub-contractors
C Employees **D** Cybercriminals

9 Networks of attackers, identity thieves, scammers and financial fraudsters are known as:

A script kiddies. **B** cybercriminals.
C hackers. **D** spies.

10 Which malware requires a user to transport it from one computer to another?

A Worm **B** Rootkit
C Virus **D** Trojan

11 How do viruses differ from worms?

9780170440943

TEST YOUR KNOWLEDGE

Software development practices

12 What is the SDLC management strategy adopted to prevent insecure code being included in a final software product?

Strategies for minimising potential risks

13 Explain two strategies that are used to minimise potential risks in software development.

Identifying software and data vulnerabilities

14 Web application attacks cannot be blocked by traditional networking security devices. Why not?

15 What is an SQL injection attack?

16 What is an XSS attack?

17 What is an XML injection attack? How is this different to SQL or XSS attacks?

18 How does a distributed denial-of-service attack (DDoS) differ from a denial-of-service attack (DoS)? Which is the more dangerous, and has the largest impact? Why?

Strategies to protect against web application risks

19 What are some tactics to defend against an SQL injection attack?

20 If no user input can be trusted, how should this be checked (validated)?

21 What are some tactics to defend against an XSS attack?

22 What is a buffer overflow attack? How could this vulnerability be prevented?

23 How can a server be defended against a DoS, or DDoS attack?

24 How is the management of security achieved with third-party software?

25 What are the limitations on this strategy?

Integrity of data

26 Why is integrity of data important?

27 What are the threats to data integrity?

28 What are the characteristics of data integrity?

29 Why could validation be unsatisfactory, and how can the reasonableness of data be tested?

SCHOOL-ASSESSED TASK TRACKER

☑ Project plan	☑ Justification	☑ Analysis	☑ Folio of alternative designs ideas	☑ Usability tests	☑ Evaluation and assessment	☑ Final submission

9780170440943

APPLY YOUR KNOWLEDGE

A recently established software development company is considering expanding its workforce and relocating to the top floor of a three-storey building.

The original directors will be joined by four others, who will be responsible for web design, database design, Python language development and sales and marketing.

Each designer will have their own office, and the web designer will also be the reception and phone-switch operator. Each worker will be provided with a notebook, which can be taken home, and a desk station with an external monitor.

The wi-fi will be shared with a friendly company that resides on the second floor.

The servers will be kept in the basement. The building has a lift.

The Internet connection will be on a basic (NBN12) plan with off peak speeds of 1Mbps upload and 12Mbps download.

DogMatch WebAPP details:

- The first software project will be a dating WebAPP that will match dog owners with a common interest in a variety of categories.
- The WebAPP will collect personal information, pictures of the dog and owner, and provide a match according to an algorithm.
- A meetup will be organised at a mutually acceptable safe location by swapping contact details.
- The WebAPP will be written in Python, uses an SQL database and will be available on any web browser.
- Members of the DogMatch WebAPP will pay a monthly subscription after entering credit card details or direct debit details into a web form and paying the initial joining fee.
- New members will be sent a confirmation email with BSB bank account details for them to arrange a bank account direct debit.
- Login to the WebAPP will be through a four-digit number passcode with a limit of 30 tries before a 30-minute timeout.

Questions

1 Design a secure physical arrangement for the software company. Include any hardware that will be necessary to ensure sensitive, and expensive, equipment and information can be retained within the building.

2 While developing the software, the Python designer had difficulty completing the final hash algorithm to meet the release deadline. The hash table worked slowly, causing delays on logins, so the developer skipped the final validation and just kept the password as plain text.

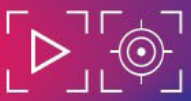
Identify some of the:

a physical security issues.

b logical (software) security issues.

c humanware security issues.

9780170440943



3 What may the implications for expansion be if the webAPP goes viral?

4 What are the implications if the webAPP gets infected with a virus?

5 Unknown to the directors, there is a security breach when one of the notebooks is stolen. The notebook does not have a login password and uses autofill to login through the browser to an Admin account.

   a What are the obligations on the directors when there is a data breach?

   b What are the liabilities for the company if bank account details are lost, with personal details, names, addresses and phone numbers, driver's licences and dog name(s)?

SCHOOL-ASSESSED TASK TRACKER

☑ Project plan	☑ Justification	☑ Analysis	☑ Folio of alternative designs ideas	☑ Usability tests	☑ Evaluation and assessment	☑ Final submission

9780170440943

Chapter

7 Software security

KEY KNOWLEDGE

After completing this chapter, you will be able to demonstrate knowledge of:

Interactions and impacts

- reasons why individuals and organisations develop software, including meeting the goals and objectives of the organisation
- key legislation that affects how organisations control the collection, storage, including cloud storage and communication of data: the *Copyright Act 1968*, the *Health Records Act 2001*, the *Privacy Act 1988* and the *Privacy and Data Protection Act 2014*
- ethical issues arising during the software development process and the use of a software solution
- criteria for evaluating the effectiveness of software development security strategies
- the impact of ineffective security strategies on data integrity
- risk management strategies to minimise security vulnerabilities to software development practices.

Reproduced from the VCE Applied Computing Study Design (2020–2023) © VCAA; used with permission.

FOR THE STUDENT

Students should be able to analyse and evaluate software development security strategies within an organisation and recommend a risk management plan to improve current practices.

FOR THE TEACHER

This chapter is based on Unit 4, Area of Study 2 and, together with chapter 6, provides the key knowledge required to complete Unit 4, Outcome 2. At the end of chapters 6 and 7, students should be able to:

- respond to a teacher-provided case study to examine the current software development security strategies of an organisation
- identify the risks and the consequences of ineffective strategies
- recommend a risk management plan to improve current security practices.

Alamy Stock Photo/Cultura Creative

9780170440943

Why develop software?

Sometimes, software needs to be written to specific criteria rather than being purchased off the shelf. Although open source software is freely available, a developer is often needed to customise the program to a particular purpose. Sometimes software is needed to perform specific tasks that off-the-shelf software cannot do. This can affect both individuals and organisations. Sometimes the goals and objectives of organisations are not being met, and software needs to be developed to meet these.

Goals and objectives of organisations

Organisations tend to go through many changes over time. These changes are generally the result of a strategic plan. A strategic plan is a process for identifying long-term goals within an organisation. For example, a school is an organisation. A school needs to establish a strategic plan outlining how it intends to maintain or increase enrolments, introduce new courses and perhaps erect a new building. This type of planning looks beyond the day-to-day running of the organisation and concentrates on future developments. These plans could range anywhere from two to 25 years.

Once an organisation has developed a strategic plan, a mission statement is developed based on the organisation's purpose, visions and values. As you read in in chapter 4, the **mission statement** is the basis for establishing a set of common goals that will help accomplish the organisation's aims. These are known as **organisational goals** (see chapter 4).

For example, a school may establish targeted goals, such as the introduction of a vocational education course within two years, or the completion of a new science and technology centre in the next five years. To achieve these goals, the organisation needs to develop a list of **objectives**. Objectives are small, achievable tasks that need to be undertaken in order to accomplish a big task. Objectives are measurable. For example, an objective may be to increase student enrolments by 10 per cent within 12 months.

FIGURE 7.1 The relationship between mission statement, vision statement, goals and objectives

Organisations are increasingly dependent on the use of software to achieve their goals and objectives. When planning the software, the systems analyst will identify a **goal**. This explains the specific role of the software in achieving the organisational goal and, ultimately, the company's mission. Setting up the right type of software can help an organisation make improvements in efficiency, effectiveness and decision-making.

Identifying organisational goals is important to understanding how and why organisations operate.

THINK ABOUT SOFTWARE DEVELOPMENT 7.1

1. Can you think of any type of organisation that would need a strategic plan? For what purpose?
2. Using the Internet, search for two examples of a mission statement.

A mission statement concentrates on the present, whereas a **vision statement** focuses on the future. A mission statement explains the company's reason for existence. It describes the company, what it does and its overall intention. A vision statement describes the organisation as it would appear in a future successful position.

Organisational goals and objectives will often relate to improving the efficiency or effectiveness of operations.

A systems analyst performs both the analysis and the design of the software. As well as being a good planner, a systems analyst must be a good communicator. As part of the analysis, the systems analyst is responsible for talking with users and translating their needs into a design, which is then passed on to programmers to build.

SCHOOL-ASSESSED TASK TRACKER

Project plan	Justification	Analysis	Folio of alternative designs ideas	Usability tests	Evaluation and assessment	Final submission

9780170440943

Minimising risk

Threats to data and information occur every day. Data can be compromised through activities such as theft (such as by disgruntled workers or criminals who will make money selling the data), loss of devices (such as accidentally leaving laptops somewhere), neglect (not erasing data when recycling computer hardware) and not following appropriate data-handling procedures and policies. Collecting, storing, sending, encrypting, gathering and disposing of data all involves risk. To protect data and information, organisations need to manage this risk by putting security measures into place, such as:

- securing computers, servers and wireless networks
- utilising anti-virus and anti-spyware protection, and firewalls
- storing data backups off-site and ensuring that backups occur routinely
- securing passwords
- ensuring staff are familiar with digital systems policies and procedures
- becoming familiar with their legal obligations.

Key legislation for storage and disposal of data and information

THINK ABOUT SOFTWARE DEVELOPMENT 7.2

Why is this legislation contentious? Why is privacy protection important?

There are several key laws relating to the information systems and telecommunications industries. At a federal level, the *Copyright Act 1968* protects the rights of creators of creative and artistic works. Other key legislation includes the *Privacy Act 1988* (see chapter 4), which governs how information about people can be used. In Victoria, we are especially concerned with the *Privacy and Data Protection Act 2014* (see chapter 4) and the *Health Records Act 2001* (see chapter 4). Combined, these laws govern the collection and use of private information by both government and non-government organisations, at both state and federal levels. Employers and government agencies have a legal responsibility to ensure that these laws are implemented within their organisations. Organisations must also make employees and customers aware of their rights and their responsibilities in relation to these laws.

This section examines the key laws affecting the storage and disposal of data and information held by organisations.

Copyright Act 1968

The *Copyright Act 1968* is a federal law that recognises that any original creative or artistic work is the property of the person who created it. Any person wishing to use another person's work must obtain permission and/or pay for a licence. The *Copyright Act* protects the creator of an original work from unauthorised reproduction, conversion, adaptation, transmission or publication of their intellectual property (IP), which includes:

- original literary, dramatic, musical and artistic works
- websites
- software
- electronically recorded music, films and books.

The last known amendment to this Act was passed in December 2017. The Act implemented a number of amendments that were supported by both creators and users of copyright content. Among these amendments were: extending the exception for exams to include online exams; allowing libraries to make 'preservation copies' of 'original versions' such as manuscripts; and simplifying and updating the provisions that allow accessible format versions to be produced for people with disabilities.

9780170440943

The *Copyright Act 1968* was amended by the *Copyright Amendment (Digital Agenda) Act 2000*, the *Copyright Amendment Act 2006* and the Australia–United States Free Trade Agreement (AUSFTA).

Copyright protection exists automatically, as soon as intellectual property is created and recorded in a way that can be seen or heard (for example, written, recorded, filmed or put online). However, the *Copyright Act* does not cover ideas, concepts, styles, techniques, information, names, titles, slogans, people or images of people. You do not have to register for copyright as you do for patents or trademarks. You do not need to use a copyright symbol or statement, although they are recommended.

You have the right to protect your own original works using technological devices such as encryption or copy protection.

Without permission from the copyright holder, it is illegal to:

- digitise a non-digital work, such as ripping a CD to MP3, or converting a DVD to an MKV file
- make or import devices or software to bypass copy protection
- remove or tamper with a copyright notice
- share copyrighted material online
- keep or share programs recorded from TV
- publish unauthorised screenshots from some web pages or software.

There are a few limited exceptions that allow IP to be used for education, review, satire or fair use. For most copyright-related criminal convictions, an individual may face a fine of up to $117 000 and/or up to five years imprisonment. An organisation may face a fine of up to $585 000. The possible term of imprisonment is up to five years. It is important to note that employees who infringe copyright by pirating software on work computers are liable, but their employers can also be required to share some of that liability.

CASE STUDY

INSIDE LAW: Dallas Buyers Club and Village cases explained

28 April, 2016 by Sonia Borella

Two of Australia's top media and entertainment lawyers – Sonia Borella and Dan Pearce from national law firm Holding Redlich – explore two recent developments in combating piracy.

Rights holders have been active recently in taking steps to counter unauthorised downloading of their films. We catch up on the final outcome in the Dallas Buyer's Club claim and review the first claim under the recently introduced anti-piracy legislation.

Enforcing rights in copyright material has its obstacles but compensation is not unattainable – lessons from the Dallas Buyers Club case.

The owners of the film 'Dallas Buyers Club' have not proceeded with their claim for payments in connection with the unauthorised downloading of the program, but the action is still an important case for rights owners.

By way of background, in October 2014, a preliminary discovery application was filed in the Federal Court of Australia (Court) by the rights holders in the film Dallas Buyers Club (Film), naming six Australian ISPs as respondents.

The purpose of the application was to obtain the details of customers associated with 4726 IP addresses that were used to download and share copies of the Film using the peer-to-peer file-sharing service BitTorrent. The intention was to then issue letters of demand to those customers for the infringement of the copyright in the Film, seeking compensation.

SCHOOL-ASSESSED TASK TRACKER

☑ Project plan	☑ Justification	☑ Analysis	☑ Folio of alternative designs ideas	☑ Usability tests	☑ Evaluation and assessment	☑ Final submission

9780170440943

On 7 April 2015, the Court delivered its decision on preliminary discovery in favour of the rights holders in the Film, and ordered that various ISPs, including members of the iiNet Group and Dodo, provide to Dallas Buyers Club LLC (or parent company Voltage Pictures LLC) the names and addresses of customers who are alleged to have used BitTorrent to share the Film.

But the order was conditional – the communications to be sent to alleged infringers needed to first be approved by the Court in order to prevent 'speculative invoicing' against alleged infringers without a proper legal basis.

Due to the nature of the preliminary proceedings, there was no decision as to whether copyright in the Film had been infringed, but the Court did 'not regard as fanciful the proposition that end-users sharing movies on-line using BitTorrent are infringing the copyright in those movies'.

The ISPs claimed in opposing the discovery that naming individual users would be economically pointless, as the compensation for the infringement would be in the order of $15.00. Representatives of Voltage Pictures LLC reportedly said that bringing proceedings against alleged infringers has value beyond the compensation recovered.

Further, in the case of multiple downloaders, the Court said it must be considered at least plausible that a copyright owner may be able to obtain aggravated damages under section 115(4) of the *Copyright Act 1968 (Cth)*. The ability to obtain aggravated damages exists partly because of the need to provide deterrence.

The rights holders in the Film proposed to demand of the alleged infringers a sum including:

a the cost of legitimately purchasing the Film
b a licensing fee relating to each infringer's uploading activities
c punitive damages, based on the alleged infringers other illegal downloading activity, and
d a claim for damages, for the costs incurred by the rights holders in the Film in bringing the application.

On 14 August 2015 the Court handed down its decision, which imposed certain limitations on and requirements of the rights holders in the Film.

Of particular concern to the Court was whether the payment demands by the rights holders were reasonable in a copyright infringement context.

The Court found that it was reasonable to demand the sum equal to the cost of purchasing the Film, given that sharing the Film online was undoubtedly copyright infringement, and reasonable compensation would be equivalent to the cost for the alleged infringer to have seen the Film legitimately.

The Court also found that the rights holders in the Film had expended significant resources and legal costs in obtaining the infringing IP addresses, and therefore that the claim for damages was also reasonable.

The Court did not regard the demands for sums equal to a licensing fee and punitive damages as reasonable.

The Court concluded that it would only order the ISPs to hand over the customer details sought if the rights holders in the Film provided the Court with a written undertaking that they will restrict the demands to those which the Court has ruled as permissible.

The Court also ordered that this undertaking be secured by the lodging of a $600 000 bond.

The rights holders in the Film then said the bond amount they would pay would only be in the amount of $60 000 for access to 472 names initially, and that the costs sought would only be for an individual licence fee as well as damages for legal costs incurred.

The Court then gave the rights holders until February 2016 to appeal the Decision, after which time the proceedings would be terminated.

The drawn-out battle came to an anti-climactic end when the rights holders in the Film decided not to proceed any further, and failed to lodge an appeal within the time limit imposed by the Court.

9780170440943

The rights owners in the Film had to overcome several obstacles but fell at the final hurdle. They had the capacity to claim compensation for the permissible demands, including the purchase price of the Film and the costs expended in obtaining the IP addresses.

However, at their election, the matter did not progress any further, which begs the question as to whether, for copyright owners, these sorts of applications can be brought cost effectively.

This case has a lesson for infringers in putting them on notice and acting as a potential deterrent.

It also provides a roadmap of sorts for copyright owners as to the best way to approach these sorts of situations, including an indication as to the types of restrictive undertakings a court may impose to prevent things like speculative invoicing taking place.

It would seem that if copyright owners are prepared to act in accordance with any condition the court may impose, that their enforcing of their intellectual property rights and the process involved in holding infringers to account for online piracy is now more transparent following this decision.

Australian Government passes Bill aimed at reducing instances of online piracy, but is legislation the answer?

In June 2015, the Australian Government passed the *Copyright Amendment (Online Infringement) Bill 2015 (Bill)*, the purpose of which is to prevent unauthorised access to certain websites, including the BitTorrent website, The Pirate Bay, and to ultimately reduce instances of copyright infringement.

For rights holders such as film and television production companies, the Bill provides a legal avenue to protect their rights, through an application being made to the Court for an order requiring ISPs to block access to websites.

Importantly, the website in question accessed by alleged infringers must have the principal purpose of copyright infringement.

While this may act as a deterrent for online piracy, it will not of itself prevent illegal downloaders from continuing to download from the websites in question. Further, in practice, the individual activity of the downloader will only be changed once the Court orders that the ISP take reasonable steps to block the website in question, following an initial application to the Court by the rights holder.

Although not necessarily a complete solution to the problem of online piracy in Australia, the Bill is a step in the right direction, and provides rights holders with a legal avenue available to protect their original copyright material.

Village Roadshow (Village) is amongst the first to utilise the newly introduced anti-piracy legislation, in recently filing an application with the Court for an order to be made to compel the ISP in question to block piracy website, SolarMovie. The basis of the claim is that this website's primary purpose is the facilitation of copyright infringement. Although these types of claims have been successful in other overseas jurisdictions, this is a test case for the Australian anti-piracy laws, and it will be interesting to see whether Village will be successful in holding the operators of the infringing website accountable for the damage they are causing to its industry.

– Dan Pearce

Where to from here?

The combination of increased avenues for Australian consumers to access television and film content legally, together with the implementation of new deterrent legislation, and possibly more cases like Dallas Buyers Club tackling this issue, will hopefully go some way to reducing the number of instances of online piracy in Australia.

© Intermedia, IF Magazine. 'Inside law: Dallas Buyers Club and Village cases explained' 28 April, 2016 by Sonia Borella. Accessed from https://www.if.com.au/inside-law-dallas-buyers-club-and-village-cases-explained/

9780170440943

Since the AUSFTA was implemented on 1 January 2005, copyright generally applies for the life of the creator plus 70 years. The copyright holder may not necessarily be the author, performer, or director if someone else (e.g. a recording company) paid for these works to be produced. Employers usually hold copyright over material that their employees create, as do film, game and music producers. Sometimes the performer may own a share in the copyright held by these organisations. Copyright can even be sold.

According to the Australian Copyright Council, 'copyright is infringed if copyright material is used without permission, in one of the ways exclusively reserved to the copyright owner'. This means that someone may not use a whole or a part of a work, including changing or adding to it, without seeking permission from its copyright owners. For example, a student producing a video for their local sporting club must seek permission to use any music or video clips if they are not the student's own original work. Similarly, someone who imports then sells copyrighted items from overseas without permission is considered to be in breach of copyright.

There have always been some provisions for 'fair use' of copyright material, such as by schools (for research or study purposes), libraries, reviewers (e.g. book and film critics) and government bodies.

When the *Copyright Amendment (Digital Agenda) Act 2000* came into effect in early 2001, the *Copyright Act* was updated to cover work published electronically. The main purpose of the amendment was to extend the *Copyright Act* to cover works that were produced, stored or transmitted digitally. This includes the use of web-based materials, digital sound and video recordings (including free-to-air broadcasts, CDs, DVDs and MP3s) and circumvention of technologically based copyright protection measures. These provisions were further extended by the AUSFTA in 2005.

In 2006, the *Copyright Act* was further updated to provide more direction for users, such as strengthening the owner's rights to their digital material. In addition, it makes provision for users to access some legitimate copyright material without breaking the law, such as time-shift recordings, although users are not permitted to store their recordings.

New exemptions relating to personal use of recorded works have allowed consumers the right to make copies of works they have purchased and transfer them into other formats for personal use. This means that it is legal to copy music from CDs that you own into an MP3 format to be used on a personal music player. People are also able to transfer tapes and vinyl records to an electronic format. Consumers are also permitted to record television and radio programs to watch or listen to at a later time (time shifted). Again, this is only for personal use, and these recordings cannot be distributed to others. Enforcement measures include on-the-spot fines and proceeds of crime remedies.

In 2015, further amendments were made to the *Copyright Act* to incorporate online infringements. The Amendment to the *Copyright Act* was intended to disrupt large-scale websites that operate outside Australia and distribute (or facilitate the distribution of) infringing material to Australian consumers. It enables copyright owners to apply to the Federal Court of Australia (the Federal Court) to block access to an online location that meets certain conditions.

Making and distributing copies of games, music and software, even if it is not for any personal financial gain, is illegal in Australia. These acts are commonly referred to as **piracy**.

Further information on Australian copyright laws, including information sheets on a wide variety of copyright-related questions, can be obtained from the Australian Copyright Council

Copyright and music, computer games and computer software

Copyright legislation means that what you can copy or reproduce in other formats is highly restricted. In terms of music, as noted previously, when you buy a CD you have the right to make a backup copy, or even to rip a copy to MP3 format to play on your personal music player. You are permitted to download music from the Internet via peer-to-peer transfers, but only if you have the permission of the copyright holder. There may be specific terms and conditions for online music stores that allow you to make a certain number of copies for personal use, but these vary between distributors.

Computer games fall into several categories because they incorporate the program code, as well as audio and video works that may be licensed from other copyright holders. Generally, you are permitted to make a backup copy of the game itself, but not any of the artistic works that may also be on the media, such as video or audio, without seeking the permission of the relevant copyright holders. You may lend a legitimate copy of a game to someone to play, but it is illegal to play an infringing copy. Under the AUSFTA, any devices specifically designed to bypass copyright protection measures are considered to be illegal.

Computer software is treated as a 'literary work' under the *Copyright Act*. As with games, you are permitted to make a backup copy of the software only, but not of any associated video or audio works unless the licence allows this. You are not, however, permitted to bypass copy protection features in order to make your backup. If your original media are destroyed, the *Copyright Act* allows you to make another backup from the first backup. Naturally, it is illegal to make multiple backup copies and distribute them to other people. The specific software licence will tell you how many times you are allowed to have the program installed simultaneously.

THINK ABOUT SOFTWARE DEVELOPMENT 7.3

1. Why have personal music players, such as iPods, necessitated a rethink about Australian copyright laws?
2. Explain what is meant by 'fair use'.
3. What do you and your classmates see as 'fair use' for music purchased online or in a shop?
4. What are the arguments for and against a 'fair use' amendment to the *Copyright Act*?

Copyright and cloud computing

Many copyright owners are using **cloud** computing services to deliver copyright material to users, such as iTunes, Spotify, Netflix and Stan. These subscriptions or pay-per-use services provide on-demand access to large libraries of properly licensed music, films, books and other content.

© Spotify; © Netflix; © Stan

FIGURE 7.2 Copyright material can be delivered to users using services such as iTunes, Spotify, Netflix and Stan.

SCHOOL-ASSESSED TASK TRACKER

☑ Project plan	☑ Justification	☑ Analysis	☑ Folio of alternative designs ideas	☑ Usability tests	☑ Evaluation and assessment	☑ Final submission

9780170440943

Conversely, individuals tend to engage with cloud computing services to store copyright material they have copied or 'ripped' themselves, such as music files copied from a CD. The advantages of storing these copies on remote servers is that it means content from multiple computers and devices, including mobile devices, can be accessed easily. The issue, however, is that the copyright holders of the material may object to this use.

Penalties for infringing copyright

We now know that the infringement of copyright includes activities such as selling or playing pirated software, games and music, decrypting television broadcasts, removing copyright protection and importing copyright material without authorisation. Most of these actions can be tried in court as civil actions. In general, copyright infringements that involve some kind of commercial dealing are criminal offences. For civil actions, the damages vary depending on the level of infringement and compensation

Cable TV broadcasters transit signals in an encrypted way so that they can charge a fee for viewing their content. Decrypting television broadcasts refers to signal theft of pay TV without permission from the original broadcaster.

Privacy Act 1988

The *Privacy Act 1988* was introduced in chapter 4 (see page 153). This chapter will investigate how the act applies to the collection, storage and communication of data.

What is included in the *Privacy Act*?

The *Privacy Act* includes the following:

- Thirteen Australian Privacy Principles (APPs) that apply to the handling of personal information by most Australian and Norfolk Island government agencies and some private sector organisations
- Credit reporting provisions that apply to the handling of credit-related personal information that credit providers are permitted to disclose to credit reporting bodies
- The collection, storage, use, disclosure, security and disposal of individuals' tax file numbers
- The handling of health information for health and medical research purposes in certain circumstances, where researchers are unable to seek individuals' consent
- An Information Commissioner who approves and registers enforceable APP codes that have been developed
- The option for a small business operator, who would otherwise not be subject to the Australian Privacy Principles (APPs), to opt-in to being covered by the APPs

Who is covered under the *Privacy Act*?

The *Privacy Act* gives an individual control over the way their personal information is handled. The *Privacy Act* allows individuals to:

- know why their personal information is being collected, how it will be used and who it will be disclosed to
- have the option of not being identified, or of using of a pseudonym in certain circumstances
- ask for access to their own personal information (including health information)
- discontinue receiving unwanted direct marketing
- ask for incorrect personal information to be corrected
- make a complaint about an entity covered by the *Privacy Act*, if personal information has been mishandled.

9780170440943

Australian Privacy Principles

As part of the *Privacy Act*, the Australian Privacy Principles (APPs) were devised to set out the standards, rights and obligations for collecting, handling, holding, accessing, using, disclosing and correcting personal information.

The Australian Privacy Principles generally apply to Federal Government agencies. They do not apply to local councils, or State or Territory Governments. Some states have their own privacy laws, such as Victoria's *Privacy and Data Protection Act 2014*.

The APPs oversee the handling of personal information by:

- Australian and Norfolk Island government agencies
- all private health service providers
- businesses that have an annual turnover of $3 million or those that trade personal information.

TABLE 7.1 The Australian Privacy Principles

APP 1	**Open and transparent management of personal information** Ensures that APP entities manage personal information in an open and transparent way. This includes having a clearly expressed and up-to-date APP privacy policy.
APP 2	**Anonymity and pseudonymity** Requires APP entities to give individuals the option of not identifying themselves, or of using a pseudonym. Limited exceptions apply.
APP 3	**Collection of solicited personal information** Outlines when an APP entity can collect personal information that is solicited. It applies higher standards to the collection of 'sensitive' information.
APP 4	**Dealing with unsolicited personal information** Outlines how APP entities must deal with unsolicited personal information.
APP 5	**Notification of the collection of personal information** Outlines when and in what circumstances an APP entity that collects personal information must notify an individual of certain matters.
APP 6	**Use or disclosure of personal information** Outlines the circumstances in which an APP entity may use or disclose personal information that it holds.
APP 7	**Direct marketing** An organisation may only use or disclose personal information for direct marketing purposes if certain conditions are met.
APP 8	**Cross-border disclosure of personal information** Outlines the steps an APP entity must take to protect personal information before it is disclosed overseas.
APP 9	**Adoption, use or disclosure of government related identifiers** Outlines the limited circumstances when an organisation may adopt a government related identifier of an individual as its own identifier, or use or disclose a government related identifier of an individual.
APP 10	**Quality of personal information** An APP entity must take reasonable steps to ensure the personal information it collects is accurate, up-to-date and complete. An entity must also take reasonable steps to ensure the personal information it uses or discloses is accurate, up-to-date, complete and relevant, having regard to the purpose of the use or disclosure.

9780170440943

TABLE 7.1 The Australian Privacy Principles *(continued)*

APP 11	**Security of personal information** An APP entity must take reasonable steps to protect personal information it holds from misuse, interference and loss, and from unauthorised access, modification or disclosure. An entity has obligations to destroy or de-identify personal information in certain circumstances.
APP 12	**Access to personal information** Outlines an APP entity's obligations when an individual requests to be given access to personal information held about them by the entity. This includes a requirement to provide access unless a specific exception applies.
APP 13	**Correction of personal information** Outlines an APP entity's obligations in relation to correcting the personal information it holds about individuals.

© Office of the Australian Information Commissioner — www.oaic.gov.au. Summary of the Australian Privacy Principles accessed from https://www.oaic.gov.au/agencies-and-organisations/guides/app-quick-reference-tool. Released under CC BY 3.0 AU, link to license: https://creativecommons.org/licenses/by/3.0/au/deed.en

Credit reporting provisions

There have been changes to the credit-reporting provisions of the *Privacy Act 1988* and to the way credit-related personal information is collected. The *Privacy Act* also encompasses a code of practice for credit reporting. The credit-reporting provisions for consumer credit include the simplification of the language used in reports, and improved privacy protections. The process to lodge a complaint has also been simplified.

Privacy and Data Protection Act 2014

The *Privacy and Data Protection Act 2014* (PDPA) was covered in chapter 4, pages 154–55.

Health Records Act 2001

The *Health Records Act 2001* was covered in chapter 4, pages 155–57.

Storing health records in the cloud

The advantages of storing health records in the cloud include the fact that this gives health care professionals access to health records and data to assist with accurate patient diagnosis and medications. The sharing capabilities of the technology also enable accurate analysis of medical histories to assist with:

- diagnosis
- minimising duplication and unnecessary testing
- long-term monitoring of chronic diseases.

The improved sharing of records also enables medical professionals to communicate and collaborate and offer a team approach to looking after the patient.

However, data breaches are a major risk of cloud computing. Health records are comprehensive and highly sensitive, making them a desirable target, especially for identity fraud, theft and blackmail. Other threats include loss of data, denial-of-service attacks and cyber attacks.

While there is protection for storing health records, complexities arise from both Commonwealth and state legislation. For example, Commonwealth Government agencies and private sector health service providers must comply with the Australian Privacy Principles (APPs) (see Table 7.1) set out in the *Privacy Act 1988 (Cth)*. The *Privacy Act* recognises

health information as a form of 'sensitive information'. However, state-based public sector agencies must comply with state- or territory-specific legislation regarding privacy, confidentiality and data management – in this case, the *Health Records Act*. Confusingly, laws vary between states and territories, and there is also significant overlap between federal and state/territory laws.

Ethical issues

Although software provides numerous benefits, negative effects, both intended and unintended, can impinge upon people's rights. Therefore, those who design, control and use software must consider the real and potential negative effects of the software, and eliminate or lessen them as much as possible. Sometimes even this may not be enough to justify the proposed collection or creation of data. It is important to take into account legal objections and ethical considerations when creating or acquiring data. The purpose for collection needs to be clear. It also needs to be articulated in the participant information statements and the consent forms provided to the people from whom information will be sought.

Ethics refers to behaving in ways that are based on our morals and accepted standards. These standards may be common in a particular society or specific to a single organisation. They apply to questionable activities over and above any legal requirements. Ethics often provide us with a set of guidelines for appropriate behaviour. Choosing to ignore these guidelines is not necessarily a crime, but it can lead to being sacked by an employer or being shunned by society. For example, the impact of violent video games on children has long been debated. Some people have voiced their concerns that video game writers should not include animated violence in their games because it can have a negative impact on children. A system of classification exists for games, similar to television and film classifications, which outlines the content that is permissible within each of the classification categories, and what content is prohibited.

© Commonwealth of Australia 2018. Australian Government: Department of the Communications and the Arts. Released under CC BY 4.0 International, link to license: https://creativecommons.org/licenses/by/4.0/

FIGURE 7.3 Classification markings ensure that consumers and parents can make informed choices.

These examples demonstrate how ethics hinge on society's values and standards. In this example, there are two competing principles. Some people would argue that protecting children from possibly harmful video games is the right thing to do. Others would argue that

SCHOOL-ASSESSED TASK TRACKER

Project plan	Justification	Analysis	Folio of alternative designs ideas	Usability tests	Evaluation and assessment	Final submission

9780170440943

it is more important to maintain freedom of expression. Often, questions of ethics become debates over which of the two principles is more important. This kind of conflict can be problematic, especially when the consequences of action are open to debate or interpretation.

Ethics are the moral guidelines that govern, among other things, the use of data collection. Often ethical principles/guidelines have an accompanying law, but the ethical principle is usually broader, and the law applies only to certain circumstances or applications of the principle.

For example, it is ethical to obtain permission to publish photos of people on websites or in promotional material. It is important to state the purpose for taking the photo and how it is intended to be used. Ethically, it is wrong to use a photo for a different purpose from that for which it was originally collected. Similarly, when using data-collection tools such as surveys, interviews and questionnaires, it is important to reassure participants that the data provided, within the limits of the law, will remain anonymous, and that their individual comments will not be able to be identified by others. This is not just to put participants' minds at rest, it is also to ensure that their privacy is in fact protected (and that non-participants in the larger group that the sample is supposed to represent are not put at unacceptable risk of suffering as a result of mistaken identification).

This may be of particular concern when deciding what must be removed to sufficiently de-identify data to protect all its potential users. It is also important when reporting personal information anonymously or using pseudonyms in a newspaper report, for example.

Ethical issues in software development and use

Working in an organisation with ethical, legal or social tensions can be uncomfortable. In particular, a lack of clarity in the policy documents an organisation uses to govern the development and use of software can lead to tensions.

The relationship between software developers and end users

A complete software development process requires the identification of all people who will or might be affected, and a risk analysis must be undertaken to address all legal, social, political and ethical issues.

Often these issues arise after the software has been used for a while. These issues can cause conflicts between developers and users. For example, there may be conflicts about:

- privacy of personal information collected
- backup copies of software
- the results of the program not meeting the expected standard
- bugs in the program
- monitoring of users' activities and sending the information back to the developers
- monitoring of work activities by the IT department
- security flaws in the software
- ergonomic issues arising from the use of the product
- the reliability of the software
- the lack of consideration given to users with disabilities
- installing advertising and other software along with the main software
- the backup system not being configured to work properly
- forcing the user to register or else the program will stop working after a time
- programs that are unsuitable for use by children

9780170440943

- use of company time and equipment for private purposes, such as email and Internet browsing
- use of impersonal voice recognition and automated answering systems.

Software developers must take into account a variety of legal obligations and ethical considerations. Some of these considerations have an effect on the development of the program, and others affect the way the program is used. Even if there are no laws already in place, there can be other considerations that could result in harm for users and others.

Intellectual property and copyright

Most software developers adhere to copyright laws. Using code developed by other programmers without paying for it, using it without obtaining permission, or modifying it but still keeping the major features can leave the programmer open to prosecution under the *Copyright Act*. This legal obligation can also apply to the content of a program, such as images, sounds and text, and the interface design. Copyright is usually enforceable in other countries through international copyright treaties.

Websites are very good sources of material that can easily be copied, often illegally. Although copying and using images, HTML and JavaScript code and Java Applets is easy to do, this should not be done without due acknowledgement of the source. Often a payment is required.

The intellectual property rights of a program do not reside with the software developer if the developer was employed by an organisation when doing the work. Therefore, code, materials and technical knowledge that is owned by the organisation cannot be used if the programmer is developing a similar program for another organisation, or for themselves. These situations are often covered by non-disclosure rules that are written into the terms or contracts of employment. For example, the employment contract might state:

> 'All intellectual property rights are vested in the employer for any inventions, new programs or new systems developed in the course of employment. Moral rights remain vested in the author or creator of any new invention/program/system or other forms of intellectual property.'

There are many websites that can be used to obtain software that has not been purchased. Downloading of this software is illegal. Programs and their accompanying crack can be downloaded. A crack eliminates copyright protection schemes such as time and number of use restrictions, need for registration, need for serial numbers and activation numbers. Cracked software and software that is illegally available for free is often referred to as 'warez'.

Some types of software are allowed to be copied often for marketing purposes, such as open source, shareware, freeware and demonstration software. Shareware, freeware and demonstration software are still copyrighted, but public domain software is not. Open-source software is a special category.

Open source

Open source refers to programs where the source code is available free for use and can be modified. Open-source code is usually created as a collaborative and combined effort by many programmers, who improve upon the code and share the changes with people in the open-source community. A larger group of programmers not concerned with ownership or financial gain will develop the program further. Programmers freely work to find and eliminate bugs in the program code and publish the result online.

Warez is used most commonly as a noun; a plural form of ware (short for computer software).

Shareware differs from open-source software, in which the source code is available for anyone to look at and make changes. Shareware is free of charge and is usually available to use for a period of time before it costs money to use. The code is not available to change. Freeware is distributed at no cost to the user; however, the source code is not made available.

THINK ABOUT SOFTWARE DEVELOPMENT 7.4

Open source produces various types of software. Can you list five open-source titles and provide a description of what each can be used for?

Open source

SCHOOL-ASSESSED TASK TRACKER

☑ Project plan	☑ Justification	☑ Analysis	☑ Folio of alternative designs ideas	☑ Usability tests	☑ Evaluation and assessment	☑ Final submission

9780170440943

The Australian Copyright Council

Copyright Amendment (Digital Agenda) Act 2000

The Australian Government has enacted changes to the Australian Copyright Act called the *Copyright Amendment (Digital Agenda) Act 2000*. Detailed explanations about copyright can be found at the Australian Copyright Council website (see weblink).

Under copyright law, computer programs are defined as literary works – 'a set of statements or instructions to be used directly or indirectly in a computer in order to bring about a certain result.' This definition can include but is not limited to pseudocode, flowcharts, a program in a book, part of a program, a website or part of a website. It can also include machine code, and various languages. Other material associated with software may also be separately protected by copyright, such as text, databases, visual art (including charts, maps and plans), video (such as instructional videos, commercials and computer video games), music and sound recordings.

In one case, the Federal Court stated that the fact that two pieces of software perform the same function does not, by itself, mean that there is any similarity between the two sets of instructions.

However, copyright does not protect the function of the program. For example, a programmer may develop a similar functioning program using a different set of instructions. Copyright protects the form in which the idea is expressed in a particular program, but not the idea for the program.

Copyright is infringed when a person uses or sells a program or a substantial part of a program without obtaining permission from the copyright owner. Licensing agreements usually specify what the licensee can do with the program. Circumventing copyright protection technology is not permitted unless it is classed as a 'permitted purpose' activity.

What you can and can't do with software is set out in the terms and conditions of your licence. There is no general right to copy software for personal use. Developers should familiarise themselves with the licences under which they distribute their software.

There are a limited number of situations that do not count as copyright infringement. In these situations, the copyright holder does not need to be contacted for permission. The basic principle of these situations is that copying does not infringe the copyright holder's ability to earn income and control the distribution and use of the program. Some examples where copyright is not being infringed include:

- making a backup copy of a computer program either to use in place of the original copy, or to store as a backup for use if the original or an earlier backup is lost, destroyed or rendered unusable
- copying software as part of a normal process of backing up files for security purposes
- making a backup copy of a computer program if there are no technological locks to prevent copying on the program
- when a programmer needs to 'pull apart' a program to find out how it works so that an 'inter-operable' product can be produced that works with the original program
- testing the program for security purposes, or for finding faults if there is no other way of doing so
- 'fair dealing' for research, study, criticism or review, news reporting and giving legal advice.

A program that is no longer available commercially still has copyright held by the owner or by whoever buys the assets of the company.

When is copyright infringed?

Copyright is infringed when a person other than the copyright owner uses a 'substantial part' of the material in any of the ways reserved for the copyright owner without their permission. For example, if software users make copies, install software on multiple devices or make

9780170440943

software available for downloading, this can infringe copyright. A person may infringe copyright by importing, selling or otherwise commercially dealing an infringing copy of computer software.

Software developers can take measures to protect their software, such as copy protection, password access and other types of technically based restrictions. There are two main types of technology prevention measures: those that confine access to the material, and those that limit or prevent copying of the material.

There are also provisions in the *Copyright Act* that give copyright owners the right to take legal action against people who make, supply, distribute or import devices to circumvent technology prevention measures.

Software developers can also protect their work by embedding details about the material, the copyright owner and related data. For example, details can be embedded in the metadata of a sound file, or watermarking and other data embedded into an image or video file. If someone tries to remove this data, a case can be made that the copyright was intentionally infringed.

Software developers should also consider the type of licence they will release alongside their software. The use of the software can range from restrictive uses to open source. The type of licence will depend on whether it is a commercial venture or if the software developer is comfortable freely distributing the software.

Software can be distributed through various app stores. Each store will have its own terms and conditions. The software developer will need to agree to these before their software can be sold on that platform. The licence will extend to revenue, acceptable software content and function, and how many copies of the app can be installed at a particular time, how many devices an app can be installed on and similar restrictions. Conversely, it is important for software users to be aware of the terms and conditions that outline how the software can be used, as well as consumers' rights in relation to factors such as making backup copies, re-downloading, and installing the software on multiple devices.

'Safe' programs

Programmers are legally obliged to produce programs that work properly. If a user can link a problem or consequence resulting from the use of a program to the work of a programmer, then the programmer will be liable for the damage caused. Program licences often state that the producer of the program is not liable if the user has used the program in a way that was not intended. However, if the program had bugs that caused a problem, the programmer may be liable.

Ethically, computer programmers must ensure that no damage or harm can come from the use of the program. This means the programmer must take into account who the users are, and must take no shortcuts in testing the program. This is an important consideration, given that programmers can write programs for all types of occupations and workplaces, including nuclear power plants, chemical factories, medical facilities and financial institutions.

Programmers have an ethical obligation to design, build and test programs that will benefit the user. Programming computers to have a beneficial impact is not easy, and is not always a conscious developmental goal. Programmers are often bound by financial and time constraints. The obligations to produce the best program possible and to remove all bugs are important. There have been cases where bridges have fallen down and planes have crashed due to programming bugs such as faulty code or badly designed features.

THINK ABOUT SOFTWARE DEVELOPMENT 7.5

Conduct an online search for 'history's worst software bugs' to find more examples.

SCHOOL-ASSESSED TASK TRACKER

☑ Project plan	☑ Justification	☑ Analysis	☑ Folio of alternative designs ideas	☑ Usability tests	☑ Evaluation and assessment	☑ Final submission

9780170440943

'Bad', hidden and malicious software

There are several illegal or unethical features that programmers can design into programs. For example, they can program the software to:

- have a *back door* so that people can bypass security features to gain access to a system
- contain hidden functions that can monitor the use of a program or a computer
- gain access to data on a computer system
- make connections to the Internet and report back to the author.

Some programs will install spyware or adware – these are usually attached to software that is installed via the Internet. The user is generally unaware of the spyware or adware, as it is attached to a program or file the user wanted to download.

Writing viruses and other such programs can also be classified as unethical. Other situations that might be considered problematic include:

- websites being programmed to store 'cookies' on computers, which record the activities of the person who visits the site
- programming games with features that challenge normal behaviour and values, such as excessive violence and other features that are discriminatory to particular genders or races
- installing camera surveillance and email monitoring, which can raise some ethical problems, especially if the employees are not informed.

CASE STUDY

Programming bugs with BIG effects

1. In 2005 a car maker announced a recall of tens of thousands of one model of its cars following reports of vehicle warning lights going on for no reason and engines stalling unexpectedly. The issue was caused by a programming error in the smart car's controlling computer code. It had a software bug.
2. In the 1980s a radiation therapy device malfunctioned and delivered lethal doses of radiation at some installations. Based upon an existing design, the Therac-25 was an 'improved' therapy system that could deliver two different kinds of radiation: either a low-power electron beam (beta particles) or X-rays. One 'improvement' was the replacement of the older Therac-20's electromechanical safety interlocks with more reliable software control. But the operating system had been written by a programmer with no formal training. Because of a subtle bug based on the speed of response of the equipment, a quick-fingered typist could accidentally trick the Therac-25 so the electron beam would fire in high-power mode but with the metal X-ray target out of position. At least five patients died and others were seriously injured as a result of this malfunction.
3. The first Internet worm infected thousands of computers in less than a day by taking advantage of a buffer overflow. The faulty code was part of the input/output network library routine called *gets()* in the Unix operating system, which is designed to get a line of text. Unfortunately, it has no limit to its input, and a large input allows the worm to take over any machine it can connect to by inputting a full program and then running it. Programmers responded by attempting to remove the *gets()* function from all working code.
4. A *Mariner 1* rocket deviated from its designated path on launch, and mission control had to destroy the rocket over the Atlantic Ocean. The investigation discovered that a formula written on paper in pencil had not been accurately copied into computer code, causing the computer to miscalculate the rocket's path. The bug had not been discovered during testing of the software.

9780170440943

The impact of ineffective security strategies on data integrity

Organisations that fail to secure the data and information they hold, and suffer losses or breaches as a result, may be subject to penalties or prosecution. For example, if private personal information is lost, damaged or exposed, organisations may be prosecuted under the *Privacy Act*. If tax records are lost, organisations may be penalised or prosecuted by the Australian Taxation Office. However, the consequences of failing to protect stored and communicated data go much further.

In some cases, data loss may mean an organisation is unable to pay wages to its staff or pay its suppliers. The need to recreate lost or damaged data, and repair or replace damaged, destroyed or stolen equipment, can result in further costs, labour and disruptions. If normal business is disrupted, the organisation will also lose income.

Data security failures may result in organisations losing their trade secrets to competitors. Depending on the level of publicity involved in the breach, the organisation may also sustain damage to its reputation, reducing customer loyalty. Its stock market value may also decline.

In 2011, Sony's PlayStation Network incurred a major security failure that became one of the most famous security breaches in history. The infographic in Figure 7.5 outlines the security failure, how it unfolded and how it could have been avoided.

The highly destructive Stuxnet worm was released in 2005, but was not discovered as a threat until 2010. The effects of the damage over those intervening five years are unknown.

After a security breach in 2009, Heartland Payment Systems (credit card processors) in the USA lost 50 per cent of their market value. They also accrued $139.4 million in breach-related expenses, including legal fees, forensic costs, fines and other settlement costs. It took them more than a year to recover from the associated stock market plunge.

FIGURE 7.4 Consequences to organisations of data security failure

SCHOOL-ASSESSED TASK TRACKER

Project plan	Justification	Analysis	Folio of alternative designs ideas	Usability tests	Evaluation and assessment	Final submission

9780170440943

Infographic World, an infographic agency. Link to website: http://www.InfographicWorld.com

Infographic World, an infographic agency. Link to website: www.InfographicWorld.com

FIGURE 7.5 The Playstation Network hack: download this infographic in full size from the NelsonNet student website

Data security

For a business, a loss of data can be anywhere from minor to fatal. For example, if a company lost records of its accounts payable, it would have no record of what had been paid and what still needed to be paid. If a retail store lost track of stock lists, it would have no way of knowing if stock levels indicated a rise in shoplifting or if staff had been making mistakes during transactions.

Failing to protect their data and information can cause businesses to lose trade secrets to competitors and can damage their reputations as trustworthy organisations. They may also face prosecution by the Australian Tax Office if their tax records are lost, as well as prosecution under the *Privacy Act 1988* if they violate the Act and personal information is lost, damaged or exposed as a result.

They may also lose income if the business cannot carry on because of the loss, and may find they are unable to pay wages. It is vital that businesses protect personal and corporate data from accidental or deliberate loss, damage or theft. This is an ongoing responsibility.

9780170440943

Just as businesses can lose their data and privacy, individuals can also lose theirs. Unfortunately, malicious behaviour such as identity theft and doxing do happen. There are also a multitude of other data security threats that, while not malicious, are nevertheless real and dangerous. However unlikely an attack may seem, it is best to employ data security measures. Data security takes two main forms: physical and software.

Doxing involves researching a person, sometimes known only by a handle (nickname or screenname) and then publishing their personal information, such as their full name, address, phone number, workplace and date of birth, online to identify them to as large an audience as possible.

Physical security

Physically protecting your data is a logical and effective first step.

- If you use a laptop or tablet and store your data on it, make sure you keep it in a secure place when it is not in use, such as in a cabinet or locked storage. Otherwise, store it out of sight.
- Do not let other people use your devices unless you know them well.
- If a friend or family member needs to use one of your devices, make sure they cannot access important data.
- Keep your doors and windows locked to prevent theft of your hardware.
- If you use a desktop computer, keep it switched off when you are not using it.
- Consider using surge-protector power outlets for all of your devices to protect the data stored on them.

Physical protection

Physical protection includes locks, guards, surveillance cameras, keys and access devices of various types such as smart cards, ID cards and biometric devices.

Human protection

Human protection involves checking the background and other aspects of people who have access to important networks and information. The worst types of security breaches are often inside jobs performed or permitted by people with access to information and networks.

Security and access factors

In some organisations, sensitive data and information of a personal, financial, military, industrial, legal or governmental nature may be stored and processed. The need for security and control of access may play a large part in the design of a system, especially the communication of the information across a network. For example, special consideration may need to be focused on the encryption of data. There have been problems with the security of credit card numbers used for online shopping because the security arrangements have been inadequate. Banks have been very careful moving to online banking because they have not been fully satisfied with the security of the financial information flowing online between the bank and the customer.

Some companies hire skilled hackers to attempt to break into their systems in order to identify security problems. Better monitoring and security hardware and software may be required as a result.

Security of and access to data need to be protected, and this should be addressed at the design phase. One technique is to make some files 'read only', so that users can access them

9780170440943

but cannot change the contents of the data. The access permissions required by various persons in the organisation need to be determined early, and should eventually be put under the control of a system manager or data administrator.

Other forms of protecting data include using:

- logon names and passwords to access different levels of the information system, and to keep unauthorised people from having access to data when there is no need for it
- encryption, which is the coding of data, so those who are unauthorised cannot read it
- biometric security methods, such as fingerprint and retina scanning
- keeping logs of activities – which files were used, when and how, and which computer was used.

Physical security also needs to be specified:

- surveillance cameras
- keys for locks
- smart cards
- tracking of staff movement with radio devices
- guards to monitor and watch staff.

Software security

Software security is extremely important to individuals and their data. Threats can come from anywhere at any time, and they can be both deliberate and accidental. Threats can come across a network and do not rely on physical access to equipment.

Use strong passwords

Ideally, passwords should be at least eight characters in length and should include a combination of numbers and both uppercase and lowercase letters. Depending on the software, you may also be able to include special characters such as punctuation. Try not to use the same password on every login. If you do, and a hacker gets into one of your accounts, they will then have access to *all* of them.

Passwords that contain common words are easily guessed, so should be avoided.

Examples of weak passwords:

- 12345
- nothing00
- password

Examples of strong passwords:

- tYjkL321pC
- n0ts4y1ng!
- w5df8k9fg

It is also a good idea not to use the same 'Secret question' information on every account. While it may be convenient to always use your mother's maiden name or the name of your first dog, if a hacker finds this out, it will be very easy for them to take over all of your accounts using the 'Forgot password' feature.

9780170440943

Shutterstock.com/Arturs Budkevics

FIGURE 7.6 Use strong passwords, and do not let other people know your password.

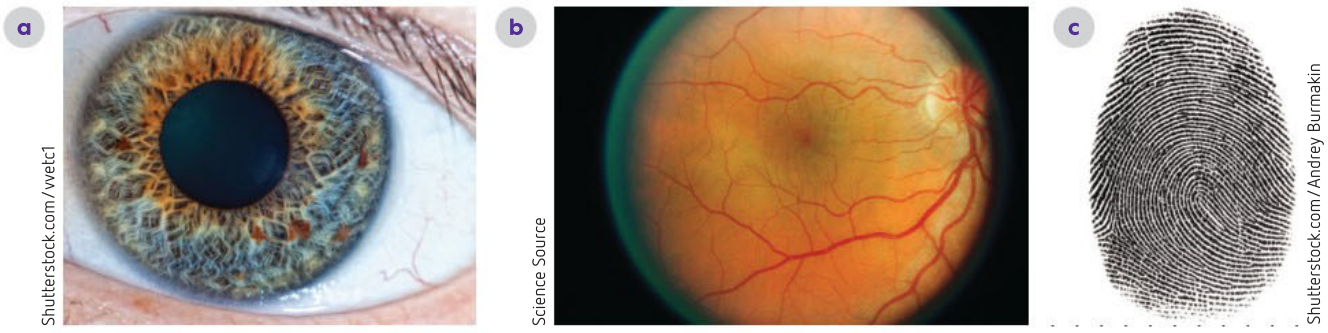
For some logins, you can use two-factor authentication. For instance, a login to my.gov.au requires a password, followed by the entry of a secret code that has been sent to the user's mobile phone, or generated using the myGov Access app. Two-factor identification relies on identifying people because they both know something (a password) and possess something (such as the linked mobile phone).

Use login passwords

You should use login passwords on your laptop, tablet, desktop computer and any other electronic device that contains personal or sensitive information. If your device is lost or stolen, you do not want someone else to be able to switch it on and immediately gain access to everything you have stored on it.

Use biometric identification

While passwords are currently the only way to control access to remote computers and resources, biometric identification can be used to control computers and resources when the user is physically present. Biometric data cannot be lost, stolen, guessed or discovered easily. Biometric signatures are unique, and include fingerprints, iris patterns (the coloured part of the eye) and retinal patterns (the blood vessels at the back of the eye). Biometric identification has long been used at Los Angeles Airport. For years, non-US citizens underwent fingerprint scans to enter the country. Now, the airport is using iris scan technology as well facial recognition technology.



a Shutterstock.com/vvetc1 b Science Source c Shutterstock.com/Andrey Burmakin

FIGURE 7.7 Biometric data: While most types of biometric data are not yet relevant to you for personal use, **a** iris pattern, **b** retinal pattern and **c** fingerprints are relevant.

SCHOOL-ASSESSED TASK TRACKER

☑ Project plan	☑ Justification	☑ Analysis	☑ Folio of alternative designs ideas	☑ Usability tests	☑ Evaluation and assessment	☑ Final submission

9780170440943

Always log out

When you are not using a computer anymore, make sure you log out and turn off the monitor. Leaving the computer logged in to your user account leaves your data vulnerable to anyone who walks past and sees that the computer is still logged on.

Log out of websites when you are not using a personal device. In fact, it is better to log out of websites even when you are using your own personal devices, but people rarely do this, which makes it easier to steal data and identities from people whose devices have been stolen.

Encrypt data

Encryption is the process of coding data so that any person who does not have the means to decode the data cannot understand it (see also chapter 3). Files on disks can easily be encrypted and decrypted using modern programs. Since it is very easy to intercept messages sent through the Internet and other networks, encryption is becoming very important for network security.

The simplest encryption method is symmetric key encryption. The sender and receiver of the coded message have the same key; a number that is used by the encryption method to encode and decode the message. The encryption works by substituting a letter for another letter. For example, if the substitution letter is the next letter of the alphabet then the key is 1; if the substitution letter was 2 behind, the key would be –2. Both the sender and receiver need to know the key.

Encryption methods come in various strengths depending on the size of the keys. When the bit length of the key is large, it is hard to break the code. A 40-bit key can generate 2^{40} = 1 099 511 627 776 different keys. A 128-bit key can generate 2^{128} = more than 3×10^{38} combinations.

Files can be secretly sent along communications lines by inserting and hiding them in another file, such as a graphics file. This is called steganography. The data is mixed with the non-essential parts of the original file and can only be separated from the original using a special program. The appropriate graphics software can still view the graphics file, and cannot detect the data that has been mixed into the file.

Previous encryption methods required a secret unlocking key to be sent with the encrypted data. If the data were intercepted, the key could also be captured and the data unlocked. Modern encryption uses **public key encryption**, which does not need a key to be sent to unlock it.

Public key encryption is the basis of SSL and TLS, which is used to encrypt the web traffic between servers and browsers. If intercepted in transit, the traffic is unreadable. Wireless signals also use public key encryption to prevent snooping and unauthorised use of wireless networks.

The bigger the numbers used for public key encryption, the harder the encrypted data is to decode. For greatest safety, choose the largest encryption key you can (128-bit or 256-bit is the current recommendation).

Use a firewall

A firewall will prevent unauthorised access to your data and information, and will deny network access to outsiders. Essentially, it will separate the Internet and other networks from the computer or LAN on which it is installed. A firewall examines the content of incoming data packets and determines whether they should be allowed to pass through.

9780170440943

Use antivirus software

Most current antivirus programs handle a whole lot more than just viruses. Many kinds of malware are around these days (see Table 7.2), so most antivirus programs are equipped to handle at least a few of these.

TABLE 7.2 Common types of malware

Malware	Description
Viruses	• Damaging code that attaches to executable files and travels with them • Payload is triggered by human actions, such as running programs • True viruses are now rare, but the name persists as a generic term for malware
Worms	• Copy themselves and travel with no human intervention or need to attach to other files • Most travel via email or over local area networks
Spyware	• Monitors the behaviour and reports browser activity to the spyware's operator • Can hijack browsers to send users to unwanted sites, or show targeted advertising based on a user's browsing history
Trojans	• Any malware that enters a system by pretending to be desirable • Often use 'social engineering' to trick people into downloading or installing them

Payload is a term that describes the destructive potential of malware. Table 7.3 lists some known malware payloads.

TABLE 7.3 Known malware payloads

Payload	Destructive potential
Keylogger	Can record users' keystrokes, including passwords, credit card information and bank account logins, and send the data to the malware operator
Distributed denial-of-service (DDoS) attack code	Makes a computer vulnerable to remote control by its operator, along with thousands of other infected machines, to participate in a DDoS attack on a remote victim. A computer that might seem sluggish to its user might actually be sending millions of information requests that can render the victim's computer unable to operate. DDoS attacks are often used to blackmail victims into paying protection money, or to attack political or religious enemies.
Adware	Inserts unwanted advertising into visited websites Not deliberately destructive, but often poorly programmed and can dramatically slow computers down or cause crashes
Spam server	Sends thousands of spam emails using the victim's computer. If discovered, innocent computer owners are identified while the spam operator remains undetected.
Ransomware	Encrypts documents on the victim's computer so they are inaccessible to the computer owner The malware operator then demands payment from the victim to receive the key to unlock their documents
Root kit	Particularly nasty malware that actively hides from the operating system and works invisibly in the background, often turning a computer into a remote-controlled 'zombie'
Deleting files or damaging operating systems	This petty vandalism, once common, is now rare. Cyber attacks are now dominated by large, organised crime syndicates and governments.

9780170440943

Even the most careful computer user cannot guard against worms or 'drive-by downloads' that can lead to infection simply by visiting an infected website. However, keep the following points in mind.

- Use a reputable, reliable anti-malware scanner.
- Always have it running and scanning opened and downloaded files for known threats.
- Keep your virus definitions up to date.
- Be aware of false positives. Sometimes a scanner can report a virus that does not exist. Free online scanners sometimes report false positives to scare users into buying their products.
- Similarly, be aware of false negatives. Some scanners may be unable to detect existing viruses. This may occur if your virus definitions are out of date, and with newly released 'zero-day' threats.

Back up your files

Data backups are the final defence against total data loss. **Backups** form an essential step in data management. Regular backups protect against a number of risks, including human error, computer crashes and software faults. Critical data files or data that are used regularly should be backed up frequently.

It is not unusual for someone to accidentally delete an important file or edit a document and later realise that some important information was removed. Maintaining a backup system can help minimise loss of data by an authorised user. Important files that have been inadvertently lost can be retrieved from the backup media.

A **full backup** copies all of the files from a device to a storage medium. This can take considerable time, and is usually performed at regular interals (such as once a week, fortnight or month). A **differential backup** copies only those files that have been changed since the last full backup. Restoration of data would involve restoring files from the full backup and then from the differential backup. An **incremental backup** is similar to a differential backup, except that it uses more than two backup media, while a differential backup uses only two media. An incremental backup only copies files that have been changed since the last incremental backup. It is the most complicated strategy from which to restore files, since it requires restoration from a full backup and then from a series of incremental backups. It is a good practice to clearly label all backup media so that you know when the backup was made and what is on it.

Location of backup files

Once you have created backups, where do you put them? Ideally, your backups should be stored in a location that is safe from theft and damage caused by extremes of temperature or disasters. Most small businesses have a fireproof and waterproof safe in which valuable company documents are stored. This might also be used to store backups. It is preferable, however, to store backups at a remote location, perhaps even in the cloud. This means that if there is a large natural disaster, such as a huge flood or an earthquake, the backups will be safe.

One last point to remember is to ensure that backups actually work when you want to restore the data. It is important to test the effectiveness of your backup files by running a disaster recovery simulation. If files cannot be restored from the backup or the system refuses to recognise them, it is better to discover this before a real emergency.

Cloud-computing companies provide offsite storage, processing and computer resources to individuals and organisations. These companies are typically third party and they store data to a remote database in real-time. The Internet provides the connection between this

9780170440943

database and the user's computer. The advantages of cloud storage include the ability to access data from any location that has Internet access, eliminating the need to carry a USB or hard drive to retrieve and store data. The ability to share files with other people and collaborate simultaneously, such as by using Google Docs, is also an advantage. Finally, if something were to happen to the computer, such as a fire or natural disaster, and the data on it was destroyed, the data would still be saved offsite in the cloud.

Google Docs allows users to upload documents, spreadsheets and presentations to Google's servers. Users can edit files using a Google application and work on them at the same time as others, enabling them to read or make edits simultaneously.

Network security

Network security should be a high priority of all network users. Network design, network management and network security are all important to the integrity and efficiency of computer systems.

Threats to networks can arise from inside or outside the organisation, and can be malicious or unintentional. They can be directed by a person or not, and can be a physical threat to equipment or a threat to software or data. Misuse by current employees for their own purposes can be a major threat to a system. Increasingly, remote access can also be a threat, as the devices are outside the major protection features of the system and may be non-standard and unprotected equipment. Table 7.4 lists some of the main threats, and ways of preventing them.

TABLE 7.4 Main threats to networks, and ways of preventing them

Threat	Type	Behaviour	Definition	Prevention	Countermeasures
Virus	Software	A program replicates within the computer and can perform malicious behaviour such as deleting files	Attached to other software and activated when software is run	Limit downloads and unauthorised software. All data and software sources need to be determined to be safe before use by scanning with a virus checker.	Installation of an up-to-date virus checker that will remove the virus
Worm	Software	A program that propagates through the network, usually by sending itself through email	A single piece of software, often an email attachment	Limit connections and use a firewall to check for suspicious activity by a program	Isolate infected computers from the network and then proceed to clean
Trojan horse	Software	Replicates within the computer and propagates through the network. Can transmit data back to sender and/or allow system access to sender.	Often part of, or attached to, a useful program but has hidden malicious parts to it	User training to detect unusual activity, especially unusual requests and activity. Use a firewall to detect unusual activity.	Installation of an up-to-date virus checker All sources need to be determined to be safe Isolate infected computers from the network and then proceed to clean Alert users to the concerns associated with downloading some types of applications

9780170440943

TABLE 7.4 Main threats to networks, and ways of preventing them *(continued)*

Threat	Type	Behaviour	Definition	Prevention	Countermeasures
Scanning	Human	Attempting to break into a system by trying different passwords	Using software to try multiple passwords until successful, or to try to guess a person's password	Set a time and limit the number of attempts to log on. Change passwords regularly.	Change all passwords and trace hacker with the use of login and activity records.
Phishing	Human	Involves sending an email to a user falsely claiming to be an established enterprise in an attempt to scam the user into surrendering private information	Deceptive attempt to obtain sensitive personal information by disguising as a trustworthy/ legitimate organisation	Don't click on URLs that are listed in emails or SMS messages.	Educate and conduct training sessions with mock phishing scenarios. Deploy a web filter to block malicious websites. Convert HTML email into text only email messages or disable HTML email messages.
Physical spying	Human	Find passwords and system information through spying	Watches the entry of passwords; often pretends to be a technician or part of organisation	Limit access to offices and staff. Use biometrics with passwords. Information needs to be encrypted.	Change all passwords, review physical security, review personal reliability, trace spy with the use of login and activity records
Spyware (also called Adware)	Software	Software that monitors the actions of a user and sends this information back to the sender and can cause advertisements to pop up	Spyware is sometimes installed as part of other software, often without the knowledge of the user	Check the fine print carefully when downloading free software. Check special sites that have information about spyware products.	Periodically scan your computer with software that detects known spyware.
Denial of Service	Human	System crashes often and network runs slow as other computers, often taken over by trojans, are activated to send multiple requests to a site or network	Overload a system so it crashes by continually sending many page requests to a site, or many requests for login or other services	Little prevention until attack occurs	Trace source of extra traffic and reject requests, and/or disconnect computer from network
Unintentional actions and technical failures					
Malfunction of hardware and software		Loss of data and equipment	Equipment breaks down	Have backup of equipment, software and data for immediate use Test software and hardware thoroughly	Have a recovery plan to replace equipment, and the reinstallation of software. Data recovery is essential.
Human error		Data loss or corruption, system misbehaves, unauthorised access	Incorrect data entered or incorrect user behaviour, or bad passwords used or not changed	Train users in correct operational procedures and limit access	Track person responsible, modify access, passwords and procedures. Use recovery plan.

9780170440943

TABLE 7.4 Main threats to networks, and ways of preventing them *(continued)*

Threat	Type	Behaviour	Definition	Prevention	Countermeasures
Physical and accidental threats					
Fire, water		Smoke, fire and water damage	Loss of hardware, software and data	Keep water supply and drainage systems away from equipment	Execute the disaster plan; extinguish the fire. Keep backups of data off-site.
Power loss and power surges		Equipment fails and may not restart	Loss of data and the inability to use equipment	Extra power lines from different sources, power surge protection	Standby power supplies, UPS system
War and terrorist attack		Physical damage	Loss of hardware, software and data	Secure facilities and keep main system off-site	Have a recovery plan to replace equipment, reinstall software and recover data

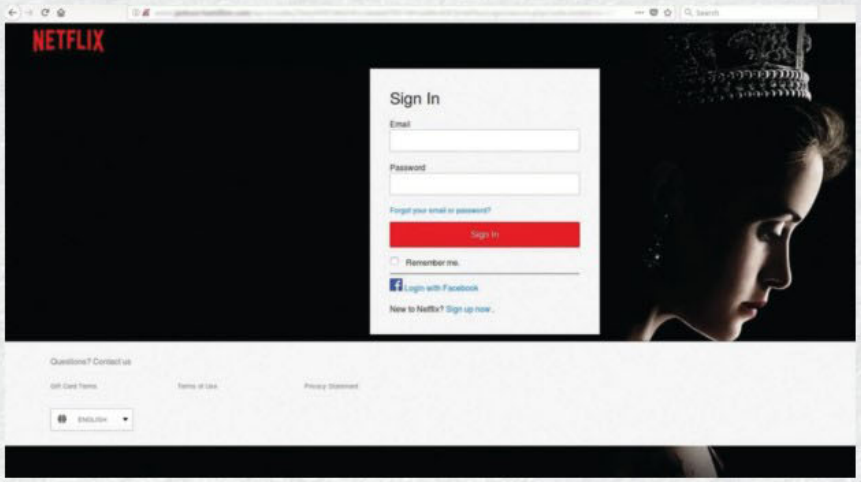
CASE STUDY

Netflix customers urged to be vigilant as 'high quality' email scam circulates

By Jenny Noyes, 29 January 2019, Jenny Noyes/SMH

Customers of streaming service Netflix are being urged to exercise caution when checking and responding to emails from the company, as a scam email that looks almost identical to a legitimate customer service message circulates.

The email purports to come from Netflix – with the spelling 'NETFLIX' using a lower-case L instead of an I, and the subject line 'We've temporarily suspended your account until you verify your details.'



© Netflix

FIGURE 7.8 Screen shot of a Netflix email phishing scam picked up in January 2019

In the body of the email, the customer is informed their account has been 'suspeneded [sic]' due to 'issues in the automatic verification process' and warned that if they don't update their details their account will be deleted.

After the customer clicks on the 'update your details' button, they are redirected to a site that looks just like a real Netflix login page, where they can 'log in' and then enter their full credit card details.

SCHOOL-ASSESSED TASK TRACKER

☑ Project plan	☑ Justification	☑ Analysis	☑ Folio of alternative designs ideas	☑ Usability tests	☑ Evaluation and assessment	☑ Final submission

9780170440943

The scam was picked up by mail security service MailGuard, which detects abnormalities in email messages and blocks fraudulent emails from reaching clients' inboxes.

People could easily go through the entire process without realising they have been swindled, said MailGuard CEO Craig McDonald, who described the scam as 'very well executed with high quality graphical elements in the email message and phishing page'.

'It's easy to imagine that it could potentially trick a lot of unsuspecting people.'

While the scam is well executed, Mr McDonald said it is not necessarily unusual – and it's not the first scam to target Netflix users in the past month.

A similar scam targeting Netflix customers with the claim their account had been 'suspended' due to issues with their billing information was detected by MailGuard's email filters in December.

The *Sydney Morning Herald* has approached Netflix for comment.

Phishing scams are the most prevalent fraud reported to the ACCC's Scamwatch, which has received more than 20 reports of Netflix-related scam emails in January so far.

'We frequently detect similar email scams that are near-perfect clones of email templates from other reputable organisations,' Mr McDonald said.

'By targeting popular brands, recipients are more likely to have a relationship with the company being impersonated. That's an instant foot in the door.'

The increasing sophistication of such email scams requires a high level of vigilance from email users and organisations alike, he said.

'Prevention has now become critical for anyone dealing with cybercrime, and organisations must implement a multi-layered defence to help minimise the risks associated with cyber security attacks.'

As for individuals, the following red flags can help to identify and avoid being sucked in by a potential scam email:

- Generic greetings, such as 'Dear customer'
- A sense of urgency: 'Ensure your invoice is paid by the due date to avoid unnecessary fees'
- Bad grammar or misuse of punctuation, incorrect spelling and poor-quality or distorted graphics
- An instruction to click a link to perform an action
- Obscure sending addresses that don't match the real company's domain URL
- If in doubt, type the web address (URL) directly into your browser rather than clicking the link, or better still, phone the company
- If you've identified a scam email, report it to Scamwatch

9780170440943

7 CHAPTER SUMMARY

Essential terms

backup making a second copy of your files

cloud Internet-based data storage and/or processing

copyright the legal right of the creator of a work to determine how that work is used

differential backup making a second copy of the data that has changed since the last backup

encryption coding data so that any person who does not have the means to decode that data cannot understand it

ethics principles of right and wrong that are accepted by an individual or a social group; ethical behaviour often guides policy makers within organisations

full backup making a second copy of your entire data set

goal an anticipated result or aim, which is specific, measurable, achievable and timely

incremental backup making a second copy of the data that was created since the last backup, using more than two media

mission statement statement setting out an organisation's purpose or what it is trying to achieve; the mission of most companies is to make a profit, while non-profit organisations tend to define their key mission as providing a service to their members

objectives small, achievable tasks undertaken to accomplish a big task

open source source code that is available free and can be modified

organisational goal how an organisation intends to go about achieving its mission

payload the destructive potential of malware

piracy the illegal making and distributing of copies of games, music and software

public key encryption encrypting data using two keys, a public key and a private key

vision statement a statement describing the organisation as it would appear in a future successful position

Important facts

1. The *Privacy Act 1988 (Cth)* was amended by the Privacy Amendment (enhancing Privacy Protection) Bill in 2012. This came into effect in 2014. As part of this Act, the Australian Privacy principles (APPs) replaced the National Privacy Principles and the Information Privacy principles.
2. As part of the Privacy Act, the **Australian Privacy Principles (APPs)** were devised to set out the standards, rights and obligations for collecting, handling, holding, accessing, using, disclosing and correcting personal information. There are 13 APPs.
3. The *Privacy and Data Protection Act 2014 (PDPA)* was introduced by the Victorian Government. It replaced the *Information Privacy Act 2000* and the *Commissioner for Law Enforcement Security Act 2005*.
4. The other key Victorian law relating to privacy is the *Health Records Act 2001*. This Act governs the collection and handling of confidential medical records.
5. **Open source** refers to programs where the source code is available free for use and can be modified. Open-source code is usually created as a collaborative and combined effort by many programmers who improve upon the code and share the changes with people in the open-source community.

9780170440943

7 CHAPTER SUMMARY

6 The *Copyright Act 1968 (Cth)* outlines the laws related to copyright. A breach of the *Copyright Act 1968 (Cth)* could result in fines or imprisonment. Computer software is treated as a 'literary work' under the *Copyright Act*.

7 Most programmers adhere to **copyright laws**. Using code developed by other programmers without paying for it or without obtaining permission, or modifying it but still keeping the major features, can leave the programmer open to prosecution under copyright laws. This legal obligation also applies to the content of a program, such as images, sounds and text, and the interface design.

8 **Copyright** is **infringed** when a person uses or sells a program or a substantial part of a program without obtaining permission. **Licensing agreements** usually specify what the licensee can do with the program.

9 Program and information system developers have to ensure that they take into account a variety of **legal obligations and ethical considerations**. Some have an effect on the development of the program and others on the way the program is used. Even if there are no laws already in place, there can be other considerations that developers need to be aware of that could result in harm for users and others.

10 Data can be **compromised** through activities such as theft (perhaps often perpetrated by disgruntled workers or criminals who will make money selling the data), loss of devices (such as accidentally leaving laptops somewhere), neglect (not erasing data when recycling computer hardware) and not following appropriate data handling procedures and policies.

11 **Threats to data and information** can be accidental, deliberate, technical and events-based.

12 There are other illegal or unethical features that programmers can design into programs, which are often called **spyware, trojans or adware**: these leave a *back door* to a program open so that people can bypass security features to gain access to a system; insert hidden functions that can monitor the use of a program or a computer; gain access to data on a computer system; or make connections to the Internet and report back to the author.

13 The writing of **viruses and other such programs**, even if not set loose on the Internet, can also be classified as unethical. Other situations that might be considered problematic include websites that are programmed to store 'cookies' on computers that record the activities of the people who visit the site, or the programming of games with features that challenge normal behaviour and values, such as excessive violence and other features that are discriminatory to women and particular races.

14 **Backups** form an essential step in data management. Regular backups protect against a number of risks, including human error, computer crashes and software faults. Critical data files or data that are used regularly should be backed up frequently. **Backups** should be regular, tested, documented and stored off-site.

9780170440943

TEST YOUR KNOWLEDGE

Review quiz

Goals and objectives of organisations and information systems

1 Explain the difference between an organisational goal and a mission statement.
2 What is a vision statement?
3 How are values important to the content of mission and vision statements?
4 Where does the purpose of mission and vision overlap?
5 Why are mission and vision statements important for organisational goals and objectives?
6 Explain why an organisation must comply with legal requirements.

Legal requirements

7 Briefly summarise the role and scope of the three key laws affecting privacy of information.
8 Why have these laws been introduced?
9 If you believe that the privacy of your information has been breached by the Australian Taxation Office, to whom can you complain?
10 What are the penalties for breaches of the *Privacy and Data Protection Act 2014*?
11 List three reasons why people illegally download files.
12 Identify a situation where downloading a file may be legal.
13 Name the Act outlining laws about copyright in Australia.
14 Define 'copyright'.
15 What is the definition of a program, according to the *Australian Copyright Act*?
16 What does copyright protect?
17 How can copyright be infringed?
18 What copying is allowed in the *Copyright Act*?
19 Explain how the concepts of copyright and intellectual property are related.
20 Outline how artists may be disadvantaged by illegal downloading and streaming.
21 Explain what 'phishing' means.
22 Name the Act that outlines the laws relating to the storage of medical records in Victoria.
23 Why are some programs called 'bad' or 'malicious'?
24 What are the privacy principles in the *Privacy Act*?

SCHOOL-ASSESSED TASK TRACKER

☑ Project plan	☑ Justification	☑ Analysis	☑ Folio of alternative designs ideas	☑ Usability tests	☑ Evaluation and assessment	☑ Final submission

9780170440943

TEST YOUR KNOWLEDGE

Data security

25 List some of the consequences of data loss for a business.

26 List four different ways to physically protect data.

27 List four different ways to limit threats to software.

28 Describe three different ways to back-up files.

9780170440943

APPLY YOUR KNOWLEDGE

The Melbourne Robotics Centre (MRC) is an organisation that runs professional designing, building and programming classes specifically aimed at school-aged children. The company offers classes after school on most weeknights and on weekends in various locations around Melbourne. Children from many suburbs participate in this extracurricular activity. The MRC is a not-for-profit organisation set up specifically to broaden children's interest in science, technology and engineering, with approximately five full-time staff and 30 casual staff. The MRC has a website, where it advertises its classes and locations. Pictures taken of students during the classes are used in the company's advertising. The website and its associated back-end database is developed by a programmer who has modified open-source software for the MRC. The MRC collects vast amounts of data on the students, such as their date of birth, home address, medical conditions, living arrangements in relation to custody, and their parents' names and occupations. The MRC relies on all its staff to update the data and its website, even though staff are not aware of all the legislation.

1. Can you identify the goals and objectives of the organisation?
2. If you were to write an organisational goal for the Melbourne Robotics Centre (MRC), what would it be?
3. Why did this organisation need to develop software specific to its needs?
4. Why does the company need to store data on their clients?
5. What key legislation should the staff be aware of specifically related to collecting, storing and communicating data?
6. What does the MRC need to do to ensure it is compliant with the *Privacy Act 1988*?
7. What measures are needed for the MRC to protect the integrity of data and information?
8. Can you identify the possible legal and ethical consequences for ineffective security practices?
9. Can you recommend a backup solution for the MRC?
10. How does the MRC protect itself from security threats?
11. Does the MRC require all the data it collects?
12. What data should be collected on the children and their parents? Apply your knowledge

SCHOOL-ASSESSED TASK TRACKER

☑ Project plan	☑ Justification	☑ Analysis	☑ Folio of alternative designs ideas	☑ Usability tests	☑ Evaluation and assessment	☑ Final submission

9780170440943

PREPARING FOR

Unit 4

OUTCOME 2

Respond to a teacher-provided case study to examine the current software development security strategies of an organisation, identify the risks and the consequences of ineffective strategies and recommend a risk management plan to improve current security practices.

To achieve this Outcome, you will draw on knowledge and skills outlined in Unit 4, Area of Study 2 – Cybersecurity: Software security. This Area of Study is covered in chapters 6 and 7 of this textbook.

Steps to follow

1. Read the teacher-provided case study.
2. Analyse and discuss the current security controls to protect software development practices.
3. Propose and apply criteria to evaluate the effectiveness of current software development security practices.
4. Identify and evaluate threats to the security of software development.
5. Identify and discuss possible legal and ethical consequences of ineffective security strategies.
6. Recommend a risk management plan and justify strategies to improve current software and data security practices.

Documents required for assessment

Submit your responses to the provided case study to your teacher. This could be in the format of written responses to structured questions, a written report or a multimedia report. (This will be determined by the requirements provided by your teacher.)

Assessment

This task is marked out of 100 and is worth 10 per cent of your study score. Your performance will be assessed using one of the following.

- → A case study with structured questions
- → A report in written format
- → A report in multimedia format

9780170440943

Index

32-bit computer systems 4
64-bit computer systems 4

acceptance testing 188
access of data, factors affecting 173–4
accessibility/accessibility testing 133, 158, 188, 192
accidental threats 245–6
accuracy (data) 133, 248–9
  clarity 248–9
  completeness 248
  correctness 248
accuracy (project plan) 202
actors 93, 94
adapt (creative design) 128
adware 274, 281
AES encryption 109
Agile software development model 144, 146–8, 224
  advantages and disadvantages 148
  applications 148
  differences from Lean software development 147
  iteration cycles 147, 148
  Kanban approach 147
  manifesto 146
  Scrum approach 147
algorithms 175
  efficiency 56–7
  for searching 53–6, 156, 175, 176–7
  for sorting 46–53, 175, 177–82
alternative execution 36
analysis phase (SRS) 91, 144
'AND' 36, 37, 38
annotated diagrams 138
antivirus software 224, 236, 281–2
Apache Subversion (SVN), case study 226
appender virus 221, 222
archiving 173
arguments 44–5
arrays 4, 5–6, 136, 174–5
assistive technology 123
association (relationship) 93
associative arrays 6–8, 175–6
asymmetric key encryption 109
Atlassian, case study 225
attractiveness (interface) 133, 143
Australia–United States Free Trade Agreement (AUSFTA) 261, 264
Australian Copyright Council 264
Australian Cyber Security Centre (ACSC) 223, 226
Australian Privacy Principles (APPs) 154, 266, 267–8
authentic data, characteristics 249
authentication 110
  multi-factor 111
  single-factor 110
  two-factor 110–11, 279
authenticity (data) 249
  challenges to 249
availability (access) 216
average-case scenarios 56, 57

back door 214, 221, 274
backups 171–2, 272, 282
  location of 282–3
bad bots 241, 242
'bad' software 274
baiting 236
bathtub curve 172
best-case scenarios 56
Big O notation 56, 57
binary files 18
binary search 53, 54–6, 178
binary selection 185
biometric data 111, 279
black box testing 188
black hat hackers 213
blogs 197
Boolean condition 36
Boolean data type 5
bot attacks
  source by country 242
  strategies to reduce the occurrence of 243–4
botnet zombies 240, 242
bots 241–2
boundary values 60, 61–3, 188, 191
brainstorming 125
branching selection 185
brute force 221
bubble sort 177–9
buckets 7
buffer overflow 243–4
built-in functions 44, 186–7
buttons and links, testing 191

C (C++ or C#) 21, 237
calculations, checking 191–2
camel case 24
case-sensitive elements 19
casting 58
CDs 172
chained conditional 38–9, 40
character data type 4–5
character encoding 4, 19
checking coded solutions, techniques for 188
checking that modules meet design specifications 58–65
Chief Information Security Officer (CISO) 212
child element 19, 93, 94, 138
cipher text data 108
clarity (data) 248–9
clarity (usability) 89, 133, 142
class visibility 46
classes 11–12, 45–6, 138
classification markings for games 269
close-ended questions 86
cloud computing 268, 282, 283
  and copyright 265–6
cloud storage 171, 172, 282–3
  health records 268
cold site recovery plan 246
collecting data 86–8
collision 7
combine (creative design) 128
combining thinking skills 124
commenting conventions 22–4
communication of message 134
compare (creative design) 128
comparison sort algorithm 179
compatibility testing 190
compiler 35
completeness (data) 248
completeness (project plan) 202
completeness (software solution) 133
component testing 190
comprehensive management plan 169
computer games
  classification markings 269
  and copyright 265
computer programs (copyright law definition) 272
computer software *see* software
concepts (project management) 77–8
  dependencies 78
  milestones 77
concise (interface) 142
concurrent tasks 79
condition test data 60
conditionals with more than one logical expression 36–8, 62
conditions 36–40

9780170440943

confidentiality (information) 215
consistency (data) 248, 249
consistency checks 18
consistent (interface) 142
constraints (software solution) 13, 91
consulting end users 126
content (functionality) (data) 248
content scraping 242–3
context diagrams 96–7, 98
  data flows 96
  drawing 97
  entities 96
  for patient information system 97
  processes 96
  Susan's music performance system 104–5
  control structures 35–43, 184–6
  statements 186
convergent thinking 123, 124
copyright 261–4
  and cloud computing 265–6
  and computer software 265, 272–3
  infringement penalties 266
  and intellectual property 260, 261, 271
  and music and computer games 265
  personal use exemptions 264
  piracy case studies 261–3
  when is copyright infringed? 272–3
  when is copyright not infringed? 272
Copyright Act 1968 (Cth) 13, 152, 260–4
Copyright Amendment Act 2006 (Cth) 261
Copyright Amendment (Digital Agenda) Act 2000 (Cth) 152, 261, 264, 272
Copyright Amendment (Online Infringement) Bill 2015 (Cth) 263
correctness (data) 248
counted loop 185
creating documents 169–70
creative design, tips for 128–30
credit reporting provisions 268
critical path 80
cross-site request forgery (CSRF) 240
cross-site scripting (XSS) 237–8
crypto mining / crypto jacking 220
CSV file 18
cyber attackers 247
cybercrime 223
cybercriminals 213
cybersecurity risks 212–56
cyberterrorists 214

Dallas Buyers Club (Film), piracy case study 261–3
data 76, 123
data accuracy 133, 248–9
data breaches 232–3, 268
  response summary 234
data collection 86–7
data dictionaries 15, 135
  as a database design tool 135
  differences between styles 136
  as a software design tool 135–6
data flow diagrams (DFD) 76, 92, 97–100
  data flows 98, 100
  data store 99, 100
  drawing 99–100
  entities 98, 100
  level 0 DFD 96, 97
  level 1 DFD 97–9, 100
  level 2 DFD 98
  processes 98, 100
  Susan's music performance system 106–8
data flows (context diagrams) 96
data flows (data flow diagrams) 98, 100
data integrity 216, 245–51
  characteristics 248–51
  impact of ineffective security strategies on 275–6
  threats to 245–7
data security 215–16, 276–86
  failure, consequences on organisations 275
data stores (data flow diagrams) 99, 100
data structures 5–12, 174–82
  searching and sorting 46–56, 175, 176–82
  types 174–6
data types 3–5
data vulnerabilities 232–6
databases
  design, data dictionary use 135
  Gantt chart for creating, case study 81–5
  standard request for records from 239
DDoS 221, 240–1, 281
debugging 58–9
decrease and conquer algorithm 54
decrypt 109
definitions 35
deleting files 173
deliberate threats 246
  detection, prevention and recovery actions 247
delimited files 18–19
delimiter 18
denial of inventory 242
denial-of-service attack (DoS) 216, 221, 284
dependencies 78
dequeue 8
design brief 12–14, 123
design consideration, for your software solution 134
design ideas
  evaluating 130–2
  generating 123–30
design phase (SRS) 75, 144
development phase (SRS) 144
dictionary 7
  see also associative arrays
differential backup 282
digital signatures 108
disaster recovery plan (DRP) 171, 246, 247
disposing of a drive 173
disposing of files 173
distributed denial of service attack (DDoS) 21, 240–1, 281
divergent thinking 123, 124
divide and conquer algorithm 50, 54
divide-by-zero errors 59
DNS cache poisoning 221
DO/WHILE loops 41–2
documentation
  internal 22–4
  software design 134–8
  using Gantt charts 80–1
documenting
  test results 193
  your testing 192–3
documents
  creating 169
  naming 170
DOM-based XSS 238
DoS 216, 221, 284
doubly linked list 10, 11
drive failure rate 172
drive reliability 172
DVDs 172
dynamic features, testing 192

economic constraints 13
effective and efficient user interfaces 141–3
effectiveness, evaluation of 199, 200
effectiveness of a solution 133, 187
  criteria for evaluating 133–4
  factors influencing 198
efficiency, evaluation of 199, 200
efficiency of algorithms 56–7
efficiency of a solution 133, 187
efficient and effective user interfaces 141–3
'ELSEIF' 38–9
encrypt 109
encryption 108–10, 227, 280
encryption algorithms 108–9
  implementing 109–10
end users
  consulting 126
  relationship with software developers 270–1
enqueue 8
entities (context diagrams) 96
entities (data flow diagrams) 98, 100

9780170440943

ethical issues 269–70
'bad', hidden and malicious software 274
intellectual property and copyright 271–3
'safe' programs 273
in software development and use 270–1, 273
evaluating design ideas 130–2
evaluating the efficiency and effectiveness of solutions 132–4
evaluation 198–200
of the client response 123
of project plans 201–2
of the software solutions 200–1
vs testing 132, 198
what it does not do 198–9
what to evaluate? 199
when to evaluate? 200–1
see also software requirements specification (SRS)
evaluation criteria 76, 132, 199, 202
evaluation methods 199, 200
event-based threats 246
detection, prevention and recovery actions 247
events 46, 80
existence checks 57, 189
expected results 58
exploit 212
exponent 4
extend (relationship) 94–5
external buttons, testing 191
external entities 96, 98, 100
external links to your product, testing 191

familiar (interface) 142
feed fetchers 241
fields 11, 176
file access, factors affecting 173–4
file management 169–74
archiving 173
backups 171–2, 272, 282–3
cloud storage 171, 172, 268, 282–3
creating documents 169–70
disposing of files 173
storing and retrieving files within a directory structure 170–1
file management plan 169
file operations 17
file organisation and storage media 173–4
file recovery 173
file sharing 283
files 17–21
naming conventions 135
storing and retrieving within a directory structure 170–1
firewalls 224, 280
fist in first out (FIFO) access 8
fit for purpose 89
floating point numbers 3, 4
flow diagrams 129
flow of execution 63
focus groups 126
FOR loops 41, 42–3
form (appearance) (data) 248–9
format checks 189
full access 110
full backup 282
function call 43
function declarations 44
function definitions 44
function visibility 44, 45, 46
functional requirements 88
Susan's music performance system 101
functionality 88, 133
functionality testing 188
functions 43–5, 186–7
pseudocode representation 45
fusion drives 174

Gane-Sarson notation style 97, 99
Gantt charts 77, 79
adjustments and annotations 194–5
for creating a database, case study 81–5
documentation using 80–1
features 79–80
time allocation resources 79–80
generalisations (relationship) 93–4
generating design ideas 123–30
techniques for 124–30
gift card balance checking 242
global variables 44
goals of organisations 148–52, 259
see also system goals
goals of the software 259
good bots 241
'good program', what makes a? 76
Google Docs 283
graphic organisers 127–8
Graphical User Interface (GUI) 60, 182
grey hat hackers 213

hackers 213
hacking 213, 275, 276
hard-coding 12
hash function 7, 175
hash tables 7–8, 175
HDD 172, 174
header comment 22
Health Privacy Principles 155, 156–7
Health Records Act 2001 (Vic) 152, 155–7, 260
breaches and penalties 156
Health Privacy Principles 155, 156–7
storing health records in the cloud 268–9
hidden software 274
high-level languages 21–2
hoaxes 221
honeynets 246
honeypots 246
hot site recovery plan 246
human error 284
human protection 277
Hungarian notation 25

IF/ELSEIF 38–9
imperfect hash 7
impersonation 247
implementation phase (SRS) 144
include (relationship) 94
incremental backup 282
indexed arrays 174–5
infinite loop 41, 42, 43, 59, 186
informal testing 190
Information Privacy Principles (IPPs) 154–5
information systems, and organisational goals 150, 151–2
inheritance 12, 46, 93, 138
inputs, inspecting 191
insecure cryptographic storage 240
insertion sort, case study 180–2
insiders 213
installation testing 190
instantiations 12
instructions 35
and syntax 182–4
integer overflow 4
integers 3, 4
integrated developer environments (IDEs) 65
integration tests 188, 190
integrity of data 216, 245–51, 275–6
intellectual property (IP), and copyright 260, 261, 271
interface 15–16
interfaces between solutions, users and networks 76, 92–108
context diagrams 92, 96–7
data flow diagrams (DFD) 76, 92, 97–100
use case diagrams (UCD) 76, 92–6
internal buttons, testing 191
internal documentation 22
conventions 22–4
International Standards Organisation, recommended specific ways of naming files and variables 135
interpreter 35
interviews 86
iteration structure 41–3, 185

9780170440943

iteration test data 60
Iterative model 145, 146

Java 21
JavaScript 21

Kanban approach (Agile development model) 147
key (associative array) 175
keyloggers 247, 281
'known-knowns, known-unknowns and unknown-unknowns' 212, 214

last in first out (LIFO) access 9
layering of zones (physical security) 217
Lean software development 147
legacy systems 201
legal requirements
  relating to ownership and privacy of data information 13, 152–7
  for storage and disposal of data and information 260–9
licensing agreements 265, 272
Likert scale 201
linear search 53–4, 177
linked lists 9, 10–11, 180
links, internal and external, testing 191
loading times 192
local variables 44
log out 280
logic bombs 222
logic errors 43, 59
logical operators 36, 37
login passwords 279
low-level languages 22
LucidChart 128, 129

mail bombing 221
maintainability 90, 202
maintenance phase (SRS) 145
malfunctions 284
malware/malicious software 212, 219, 220, 221–3, 247, 274
  payloads 281
  security procedures 224
  types of 281
man-in-the-middle attacks (MITM) 221, 233–5
managing files 169–74
Melbourne Robotics Centre (MRC) 290
memory leak 59
mental imagery 129
methods 46, 186
milestones 77
mind mapping 126–7
minimum viable product (MVP) 147, 148
mission statement 149, 151, 259
mobile site recovery plan 246
mock-ups 15–16, 138–9
module testing 188
MTBF (Mean Time Between Failures) 172
multi-factor authentication 111
multi-level inheritance 138
multiple branching selection 185
music, and copyright 265

naming conventions
  for files and variables 135
  in programming source code 24–5
naming documents 170
nested conditional 40
nesting 186
Netflix customers, email scam, case study 285–6
network security 283
  main threats and ways of preventing them 283–5
node 10
non-functional requirements 88–90
  Susan's music performance system 102–3
non-technical constraints 13
Notifiable Data Breaches (NDB) scheme 232
numeric data type 3–4

object descriptions 15, 137–8
objective results (evaluation) 199
objectives 123
  of organisations 149, 150, 259
objects 12
observations 87–8, 129–30
office environment, security 218
official goals 149
one change at a time, tested 188
online banking 219–20
online piracy 261–3
open source 271
open-ended questions 86
Open Web Application Security Project (OWASP) Testing Guide 229–30, 232
operating system filename limitations 169
operative goals 149
'OR' 37, 38
organisational goals 148–9, 259
  assisted by information systems 151–2
  common 149–50
  influence on type of information system required 150
organisations, consequences of data security failure on 275
OWASP Testing Guide 229–30, 232
ownership of data information, legal requirements 13, 152–7

parameters 44–5
parent element 19, 93–4, 138
pass by reference 44
pass by value 44
password dictionary 221
passwords
  login passwords 279
  strong and weak 278–9
patches 59
patient information system
  context diagram 97
  DFD level 1 100
payload 281
penetration testing (pen test) 232
perfect hash 7
performance testing 188
Perl 21
pharming 236
phishing 220, 236, 285–6
PHP 21, 237
physical and accidental threats 285
physical protection 277
physical security 216–18, 277–8
  indicative zone names and definitions for physical access 217
  layering zones 217
  secure building with IT resources 218
piggyback entry 218, 236
piracy 264
  case studies 261–3
  Government passes bill aimed at reducing online piracy 263
plain text data 108
plain text files 18
PlayStation Network hack 275, 276
PMI 127
pointer 10, 35
pop 9
portability 90
post-test loop 185
power losses/surges 285
pre-test loop 185
predecessor tasks 79
pretexting 236
price scraping 242
priorities quadrant 197
Privacy Act 1988 (Cth) 152, 153, 232, 260, 266–8, 276
  application 153–4
  Australian Privacy Principles 154, 266, 267–8
  credit reporting provisions 268
  and health information 268–9
  penalties 154
  what is included? 266
  who is covered? 266
Privacy Amendment (Notifiable Breaches) Act 2017 (Cth) 232
Privacy and Data Protection Act 2014

9780170440943

(Vic) 152, 154, 260, 267, 268
Information Privacy Principles 154–5
penalties 155
privacy of data and information, legal requirements 13, 152–7, 266–9
private key 109
private visibility of a function 45
problem-solving methodology (PSM) xiii–xv, 76
process (context diagrams) 96
process (data flow diagrams) 98, 100
processes (project management) 77, 78–85
documentation using Gantt charts 80–5
sequencing 79
task identification 78–9
time allocation resources 79–80
programming bugs, case study 274
programming languages 21–2
choice of 182–3
processing features 35–46, 182–7
programming source code, naming conventions 24–5
project logs 195
conditional formatting 195–6
template 196
value of 196–7
project management 77–85
concepts 77–9
processes 77, 78–85
review report 202
project plans
adjustments to tasks and timeframes 194
annotations 194–5
efficiency tips 197–8
evaluation 201–2
keeping logs 194–7
prolog 19
protected visibility of a function 45
Protective Security Policy Framework (PSPF) 216
pseudocode 6, 16–17, 139–40
array and non-array versions 175
binary search 55, 178
chained conditional control structures 39
common keywords and symbols used in 139
conditional control structures 40
converting to real code 49
discount algorithm with/without logic error 64, 65
insertion sort 181
linear search 54
of an object and a method 46
public key encryption 109
quick sort 53, 179
representation of a function 45
rules 17
selection sort 49
sequence of instructions 35
symmetric key encryption 110
to check an age range 61
types of instructions 35
WHILE loop 63
public key 109
public key encryption 109, 280
public visibility of a function 45
push 9
Python 21

qualitative data 86
quantitative data 86
queues 8–9
quick sort 47, 50–3, 179–80
quid pro quo attacks 236

RAID file storage 172
RAM 12
range checks 58, 189
ransomware 213, 220, 228, 281
readability 133, 202
checklist 191
reasonableness checks 189, 249–50
recording the progress of projects 194
adjustments to tasks and timeframes 194
annotations 194–5
efficiency tips 197–8
evaluating project plans 201–2
keeping logs 194–7
records (data structures) 11, 176
recursive algorithm 52, 54
reflected attacks (XSS) 238
relationship (use case diagram) 93
association 93
generalisation 93–4
include and extend 94–5
relevance (data) 134, 250
reliability 89–90
REPEAT/UNTIL loops 41, 43
report formats 133
reports, as part of data collection 86–7
representing designs 14–17
research (creative design) 129
responsive (interface) 142
restricted access 110
restricting URL access 244
retrieving and storing files within a directory structure 170–1
return value 43
risk appetite 231
risk minimisation 260
risk taking, persistence and bravery (creative design) 130
risk tolerance 231
robustness 90
root element 19
rootkit 221, 281
Rumsfeld, Donald 212, 213
runtime errors 42, 59

'safe' programs 273
scam statistics 223
Scamwatch website 223
scanning (threat) 284
scope creep 92
scope of the software solution 13–14, 90–1
script kiddies 213
scrubbed data 173
Scrum approach (Agile development model) 147
scrums 148
search algorithm (searching) 53–6, 175, 176–7
secure building with IT resources 218
Secure Sockets Layer (SSL) 109
security
and access factors 277–8
data integrity 245–51
data security 215–16, 275, 276–86
effect of ineffective security strategies on data integrity 275–6
network security 283–6
physical security 216–18, 277–8
software acquired from third parties 244–5
software and data vulnerabilities 232–6
and software development practices 224–8
software security 218–24, 278–83
strategies for minimising potential risks 229–32
strategies to protect against web application risks 237–44
of the web 212
what are the likely threats? 214
who are the attackers? 213–14
security attacks 218
detection and prevention 223–4
how do these attacks occur? 219–21
malware 212, 219, 220, 221–3
security clearance levels 215
security considerations 108–11
security software development life cycle (secSDLC) 224
selection sort 47–50, 180
selection structure 185
semantic errors 59
sequence/sequential structure 35, 184–5
sequencing (processes) 79
server room, locked 218
session hijacking 243
shareware 271
shoulder surfing 247
signed integers 3, 4
significand 4
single-factor authentication 110

9780170440943

SIngle source of truth (SSOT) 171
singly linked list 10
slack time 79
sleep on it (creative design) 129
smishing 247
snake case 24
sniffers 221
social constraints 13
social engineering attacks 236, 247
  protection against 236
software 76
  acquired from third parties 244–5
  copyright 265, 271–3
  distribution with app stores 273
  open-source 271
  reasons for developing 259
  'safe' programs 273
software auditing 230–1
software design documentation 134–8
software developers 76
  ethical issues 273
  and intellectual property 271
  protection of their work 273
  relationship with end users 270–1
software development and use, ethical issues 270–1, 273
software development life cycle (SDLC) 144–9, 224
  Agile development model 144, 146–8, 224
  security testing strategies 229–30
  spiral development model 144, 148–9
  Waterfall development model 144–6, 224
software development practices 224
  how to protect software and data? 225–8
  strategies for minimising potential risks 229–32
software licence 265, 272, 273
software requirements specification (SRS) 76, 86, 91–2, 123, 130, 200–1
  analysis phase 91, 144
  collecting data 86–8
  constraints 13, 91
  creating 92
  design phase 75, 144–9
  functional and non-functional requirements 88–90
  interfaces between solutions, users and networks 76, 92–101
  scope 13–14, 90–1
  software to track music performances, case study 101–8
  testing phase 144, 145
software security 218–19, 278–83
  incident/attack types across businesses 219
  security attacks 219–24
  technical threats 221
  use strong passwords 278–9
software solution
  design considerations 134
  efficiency and effectiveness 132–4, 187
  evaluation 200–1
  factors influencing design 140–1
  project management 77–85
  security considerations 108–11
  testing your 191–2
  usability 189–90
  what is it? 76
software solution specifications 123
software testing processes 188
software to track music performances, case study 101–8
software updates 227–8
software vulnerabilities 232–6, 245
solution design, factors influencing 140–1
solution requirements (design brief) 13
sort algorithm (sorting) 46–53, 175, 177–82
spam 241, 247, 249
spam filters 224, 247
spam servers 281
spear phishing 221, 236
spider diagrams 127, 128
spies 213
spiral software development model 144, 148–9
  advantages and disadvantages 149
  applications 149
split infection virus 221, 222
spoofing 221
sprints 147, 148
spying, physical 284
spyware 236, 274, 281, 284
SQL 21
SQL injection 238–40
SSD 172, 174
stable sort 180
stacking 186
stacks 9–10
statements 35
storage media, and file organisation 173–4
stored attacks (XSS) 238
storing and retrieving files within a directory structure 170–1
strings 4
strip the problem right back to its most basic parts (creative design) 128
subjective results (evaluation) 199
subroutines 186
subscription software 244–5
substitute (creative design) 128
Subversion, case study 226
successor tasks 79
surveys 86
Susan's music performance system, case study 101–8
Swift 21
swiss cheese virus 221, 222
switch/case conditional 40, 185
symmetric key encryption 108–9, 280
syntax 182–4
syntax errors 58–9
system boundary 95
system goals 151–2
system testing 190
systems analyst 258

tailgating 218, 236
task identification 78–9
technical constraints 13
teenagers, and consent 156
test cases 58, 59, 60
test data 38, 58, 59–60, 63, 190–1
  boundary values 60, 61–3, 188, 191
test results, documenting 193
testing 144, 145, 149, 189–94
  documenting 192–3
  types of 188, 190
  usability of solutions 189–90
  vs evaluation 132, 198
  vs validation 182
testing phase (SRS) 144
testing table 192, 193
testing your software solution 191–2
text files 18
threats
  categories of 214
  'known-knowns, known-unknowns and unknown-unknowns' 212, 214
  to data integrity 245–7
  to network security 283–5
time allocation resources 79–80
Time-based One-time Password (TOTP) algorithm 111
timeliness 133, 251
tolerant interface 143
trace tables 38, 58, 63–5
Transport Layer Security (TLS) 109
trojans 222, 281, 283
truth tables 37–8, 60, 62–3
two-factor authentication 110–11, 279
two-factor recovery process 247
type checks 58, 189

unacceptable security risks, determining 231
undo function 143
unified modelling language (UML) 76, 92
Unit 3 Outcome 1, preparing for 74
Unit 3 Outcome 2, preparing for 165–6
Unit 4 Outcome 1 (U4O1), preparing for 209–10
unit tests 188
universal design 123
unsigned integers 3, 4

9780170440943

uptime 90
URL access, restricting 244
usability 89, 134
  of software solutions 189–90
usability constraints 13
usability testing 190, 192–3
  conducting 193
  planning 192
use case diagrams (UCD) 76, 92–6
  actors 93
  drawing 95–6
  relationship 93–5
  Susan's music performance system 103–4
  system boundary 95
  use cases 88, 91
use cases 88, 91
user acceptance testing 188, 190
user authentication 226–7
user-defined functions 186
user experience (UX), characteristics 141–3
user interface (UI) 141
  what is a 'good' UI? 141–3
username and password 110

V-model 145
validation, vs testing 182
validation checks 188–9
validation techniques 57–8, 188–9
validation test data 59–60
variables 3, 35, 135, 136
  naming conventions 135
VCAA programming requirements 184
version control 22, 171, 188, 225
Village Roadshow, piracy case study 263
virus damage repair 213
viruses 221–2, 274, 281, 283
vision statement 151, 259
Visual Basic 21
visualisation 129

WannaCry cryptoworm 228
warez 271
warm site recovery plan 246
Waterfall software development model 144–6, 224
  advantages and disadvantages 145
  applications 145
  modified models 145–6
  phases 144–5
WBS diagram 78, 79, 81
web application risks 237–44
web scraping 241, 242–3
weblogs 194, 197
what the solution will do (scope) 14
what the solution will not do (scope) 14
WHILE loop 41, 63
white box testing 188
white hat hackers 213
whiteboards 125
wiped data 173
work breakdown structure (WBS) 78–9
worms 222, 281, 283
worst-case scenarios 56, 57

XML element types and characteristics 19
XML files 19–21
XML injection 241
XML tree 19
XPath injection 241

Yourdon-DeMarco notation style 97, 99

zero-day attacks 212
zombies 240, 242, 281

9780170440943

Table of Contents

Preface 6
About the authors 7
How to use this book 8
Outcomes 10
Problem-solving methodology 14
Key concepts 17
Unit 3: Introduction 18
Chapter 1: Introduction to programming 19
Chapter 2: Development and features of a computer program 51
Chapter 3: Software analysis 92
Chapter 4: Software development: software design 139
Unit 4: Introduction 184
Chapter 5: Software development and project evaluation 185
Chapter 6: Cybersecurity risks 228
Chapter 7: Software security 275