SOFTWARE DEVELOPMENT: CORE TECHNIQUES AND PRINCIPLES 4TH EDITION

a text for VCE Software Development Units 3 & 4

ADRIAN JANSON

ISBN: 978-0-9873914-3-8

Janson, Adrian. SOFTWARE DEVELOPMENT: Core techniques and principles 4th edition, a text for VCE Software Development Units 3&4 Adrian Janson Publishing

Software Development: Core Techniques and Principles

4th Edition

A text for VCE Software Development Units 3&4

Adrian Janson

The software development industry is an exciting and challenging one. Software developers regularly deal with complex issues such as code optimization, effective user interface design, working with the myriad of devices on the market and ensuring that the products of their efforts are error free and work to specification.

Many perceive the role of the software developer as one that is both solitary and mechanical. This could not be further from the truth. Software development is an art. It is a highly creative field in which programmers strive for innovative solutions to problems that range in complexity, scope and application. It is a rare project that is undertaken by a single software developer as many projects are large enough in scale to require teams of programmers working on aspects that need to be integrated at a later stage. There are of course a number of stakeholders involved in the software development process – the key one being the client. A good software developer needs to have communication skills that will enable them to marry the client's needs to the product and a mechanism by which the solution can be refined.

This text has been written to support the VCE Software Development Units 3&4 course from 2020-2024. The text follows the study design closely and explains the process of undertaking the Problem-Solving Methodology – which can be used to develop a solution to a specified problem. This text has been structured to follow the Areas of Study. At the beginning of each chapter, you will find a list of the Key Knowledge dot points that are covered as well as a diagram showing where the chapter fits into the course as a whole. This text contains questions to test your knowledge as well as practice exam questions from each of the Areas and Study and is designed to be the comprehensive resource for all students undertaking VCE Software Development.

ADRIAN JANSON PUBLISHING

Published in 2020 by

Adrian Janson Publishing Pty Ltd
ATF Brotto Janson Family Trust
ABN 81507629548
PO Box 2098
Mount Waverley, Vic. 3149
Australia
E-mail: janson.adrian.a@gmail.com

ISBN 978-0-9873914-3-8 (print)
ISBN 978-0-9873914-4-5 (eBook)

NATIONAL LIBRARY OF AUSTRALIA

A catalogue record for this book is available from the National Library of Australia

CONTENTS

Software Development: Core Techniques and Principles 4rd Edition

For Dad.

Chapter 1
Approaches to Problem Solving

The chapter covers Unit 3: Area of Study 1 key knowledge:

- *The stages of the problem-solving methodology*
- *Approaches to problem-solving*
    - *KK1.3 Methods of documenting a problem, need or opportunity*
    - *KK1.4 Methods of determining solution requirements, constraints and scope*

Key terms: information problem, Problem Solving Methodology, analysis, design, development, evaluation, solution requirements, constraints, scope, coding, validation, testing, documentation, efficiency, effectiveness, functional and non-functional components.

Study roadmap

	Unit 3		Unit 4	
Ch	Area of study 1 – Programming	Area of study 2 – Analysis and Design	Area of study 1 - Development and Evaluation	Area of study 2 - Cybersecurity: software security
1	✓			
2				
3				
4				
5				
6				
7				
8				
9				
10				

Solving information problems

There is an art to solving information problems. All of you will have an appreciation of this, having used software that addresses particular problems in different ways – with varying success. There is a perception amongst non-programmers that programming is a very mechanical process. The truth is the exact opposite. The process of creating a solution to an information problem is one that involves high levels of creativity and often imagination. The methodology known as the Problem-solving Methodology gives a framework to this process.

The Problem-solving Methodology (or PSM) is central to the study of VCE Software Development. Within the discipline of Software Development, many similar methodologies exist. For the purposes of this course, the methodology that we are going to study is the PSM as defined in this chapter and in the VCAA VCE Applied Computing Study Design.

An important consideration when beginning to examine the PSM as stated in the Study Design, is that of context. The PSM has been created with a view of supporting all of the VCE studies. With this in mind, its activities are stated in a general sense, and in some cases, may contain examples that are specific to one VCE unit and not another. Our discussion will place the PSM into a VCE Software Development context.

The Problem-solving Methodology (PSM)

The Problem-Solving Methodology is a process by which a solution can be found to an information problem that exists within an

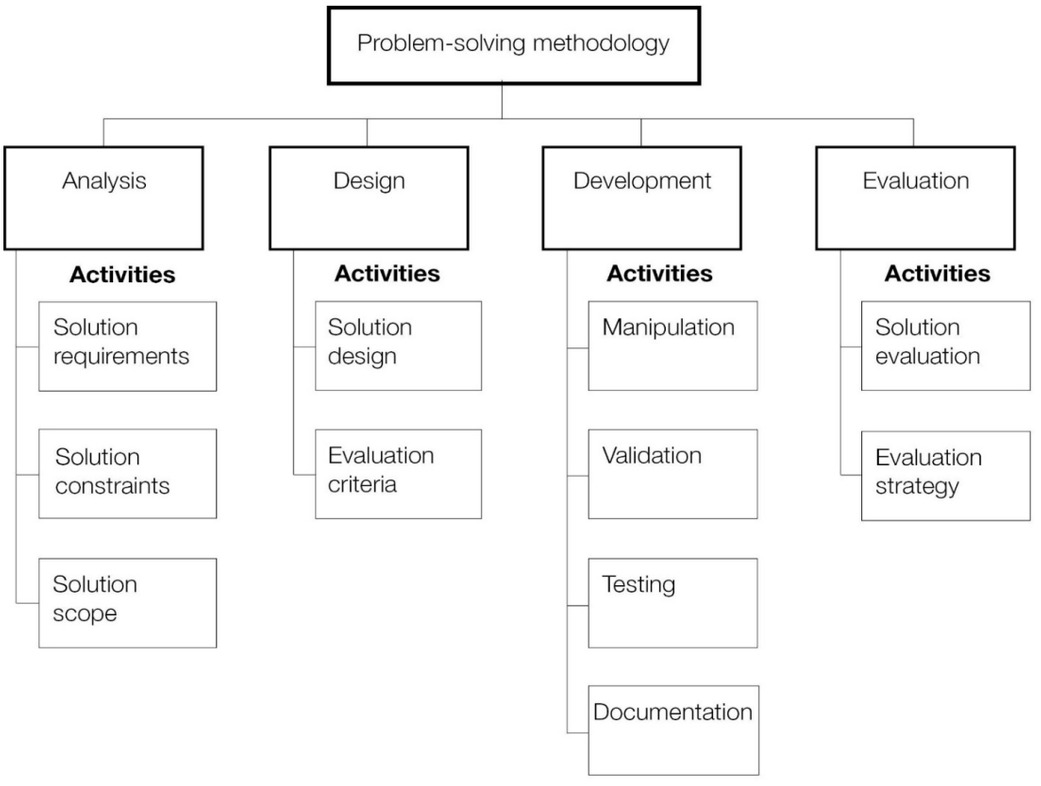


Figure 1: The Problem-solving Methodology (PSM)

information system. A software developer will follow this process to ensure that a design addresses the problems that exist, performs within specifications, integrates if needed with the existing system and can be achieved within budget and on time. We will define four stages, each with a number of activities within them.

The four stages we define are:

1. Analysis
2. Design
3. Development
4. Evaluation

It is important to note that the PSM is not necessarily a linear process. In reading the stages and sub-stages in Figure 1, the order is fairly clear. While a project would start out executing the stages in this order, and would aim to proceed from one to the next without any deviation, this does not normally occur.

The way that the PSM is implemented is known as the 'development model' and the different choices that are available will be discussed in detail in Chapter 4. For now, let's just become familiar with the stages and activities associated with each stage.

The first stage of the PSM is analysis.

1. Analysis

The analysis stage is where the information system is examined to determine what problems exist or how new elements can be added. There are a number of tools that can be used to analyse a system, but it is important to remember that an information problem involves users, processes, equipment and data and all of these must be considered. Analysis is often about asking targeted questions; together the answers help to build a picture of what is required. Analysis involves three main activities in preparing a solution: determining

the solution requirements, identifying the solution constraints and determining the solution scope, all of which are influenced by the needs of the stakeholders (any parties that have a valid link or interest in the system or information problem). These activities are often documented in the form of a Software Requirements Specification (SRS) which is a type of report that is used by software developers to document the analysis of an information problem and to enable the design stage to begin.

Systems thinking: A holistic approach to the identification and solving of problems. Systems thinking involves analysing the interactions and interrelationships between components of individual information systems (data, processes, people and digital systems), to identify how they are influencing the functioning of the whole system. This approach enables students to understand whole systems and work with complexity, uncertainty and risk.

VCAA VCE IT Study Design 2020-2023 Glossary

Determining the solution requirements

In determining the solution requirements, the key question that needs to be asked is: What does the solution need to provide?

Any given information problem will contain aspects that will be well understood and others that will not. So what is the best way to gain a thorough understanding of the problem in order to work out what is required of the solution?

A number of tools can be used to represent the information problem and come to a better understanding of its particular aspects. Tools such as context diagrams, data flow diagrams (DFDs) and use case diagrams (UCD) can be used in this way, and often all three together will be used to build a complete picture of what is occurring. Collectively, these tools clearly identify the data that is currently being gathered, how it is being used and by whom, and what information is being produced. By examining the information problem in this way, it can be easier to find out what additional data will be needed to produce the software solution that is required and what functions the solution needs to provide.

The requirements of a software solution can be classified as either **functional** or **non-functional**. Functional requirements are directly related to what the software solution is required to do. Non-functional requirements are related to the attributes of the software solution, such as user-friendliness, response rates, reliability, portability, robustness and maintainability. For example, the digital blood pressure monitor shown in Figure 2 performs two very specific functions (taking a person's blood pressure and their pulse). These are the functional requirements of the software that is running on the device. However, the software has a number of non-functional requirements relating to the way in which the information needs to be presented and the way the device responds. The device displays the pulse as a flashing heart icon and the blood pressure is displayed in a timely manner (usually less than a minute). The monitor is very easy to use with a start and stop function (user friendly) as well as a recall function that immediately displays the last blood pressure reading (response rate).

Figure 2: Digital blood pressure monitor

Identifying the constraints on the solution

Key to the success of any software solution, will be an understanding of the constraints placed upon it. Probably the most obvious constraint is cost and this will vary widely based on the size of the organization and the scale of the project. Other constraints that need to be taken into account are: speed of processing required (or available), the requirements that users have of the solution, legal requirements, security required (or imposed), compatibility with existing hardware and software within the system, the level of expertise of the users, technical support staff and the software developers themselves, the capacity of the existing system and the availability of equipment. This is by no means an exhaustive list and sometimes the constraints that are placed on a proposed software solution are hard to predict and particular to the organisation or environment.

Determining the scope of the solution

Similar to constraints is the topic of scope. The scope defines what the boundaries of the software solution will be. It also identifies what the solution will do, what it won't do and what particular benefits there will be to users. Benefits are often stated in terms of efficiency and effectiveness.

What is the difference between efficiency and effectiveness?

Two terms that are important to understand are efficiency and effectiveness. Efficiency can be measured by examining factors such as the time it takes to complete common tasks, the cost of maintaining the system and the effort required to produce the required information. Effectiveness can be measured by examining whether the goals of the system have been met, that is, how accurate the solution is.

Effectiveness: A measure of how well a solution, information management strategy or a network functions and whether each achieves its intended results. Measures of effectiveness in a solution include accessibility, accuracy, attractiveness, clarity, communication of message, completeness, readability, relevance, timeliness, and usability. Measures of effectiveness of an information management strategy include currency of files, ease of retrieval, integrity of data and security. Measures of effective networks include maintainability, reliability and the security of data during storage and transmission.

VCAA VCE Computing Study Design 2020-2023 Glossary

Efficiency: A measure of how much time, cost and effort is applied to achieve intended results. Measures of efficiency in a solution could include the cost of file manipulation, its functionality and the speed of processing. Measures of efficiency in a network include its productivity, processing time, operational costs and level of automation.

VCAA VCE Computing Study Design 2020-2023 Glossary

2. Design

Once the analysis is complete, the design of the software solution can begin. If a software developer is new to a project at this stage in the PSM, they will be provided with an SRS which outlines all of the important aspects from the analysis stage. The format and content of an SRS will be discussed in detail in Chapter 5.

The design stage consists of two main activities: planning how the solution will function given the requirements (the solution design) and determining the criteria that will be used to evaluate the solution.

Planning solution functionality and appearance (the solution design)

Designing how the solution will function is a complex activity for a software developer. It involves working out how the data that is required will be named, structured, validated, manipulated and stored. A number of tools can help to accomplish this task. Data dictionaries, data structure diagrams, pseudocode and object descriptions can all be used to do this. This activity also involves showing how the various components of the solution relate to one another. Tools that can help to accomplish this are data flow diagrams, context diagrams and use case diagrams. Lastly, it is important to be able to design how information will be presented in the software solution. This can be represented by using tools such as annotated diagrams and mock ups. Each of these tools will be explained in the following chapters.

Determining the evaluation criteria

It may seem out of place in the design stage to be considering how the software solution will be evaluated, however, this is vitally important to the success of the project. By translating the requirements of the software solution as set out in the SRS to a number of evaluation criteria, software developers have a better sense of how the success of the solution will be ultimately judged.

3. Development

The development stage of the PSM is the stage during which the software solution is built. It consists of four main activities: coding (manipulation), validation, testing and documenting. These activities do not necessarily occur in this order, as will be explained.

Coding (manipulation)

This activity in the PSM is titled 'manipulation' but for our purposes, the development stage begins with coding.

Coding a software solution is probably seen as the largest activity from the perspective of a software developer. In a sense this is true, but it is a task that is made much easier if due diligence has been given to the analysis and design stages. While coding a software solution, it is important that a software developer follows good coding conventions for the naming and structure of the code and includes ample internal documentation. While following these sorts of conventions may not affect the final product (and will be invisible to all but those involved in the development now and in the future), not doing these things leads to sloppy and hard-to-read code (and is seen as quite unprofessional). The coder writes code in an appropriate programming language in accordance with the plan, then tests, debugs and modifies the code as required.

Validating

Validating data is the process of determining the reasonableness of the data. The amount of validation that is included and the way in which it operates can have a profound impact on the effectiveness of the entire software solution. As well as preventing errors from occurring, the process of 'trapping errors' can also be used to determine if data is reasonable.

Validation really begins in the design stage. It is in the design stage that important decisions about how data will be collected, processed and output will be made. Validation is also strongly influenced by user interface design. The actual validation coding takes place in the development stage.

Testing

Testing is often an ongoing activity during the development of a software solution, and as a programmer adds elements to the program, they will test them to see that they are working and modify or fix them as needed. However, the formal activity of testing a software solution is usually conducted at the conclusion of the development of the software and is done using an exhaustive grid which covers

both the valid and invalid possibilities of the software's use.

When undertaking this task, the first step is to list all of the tests that will be undertaken. This list can be quite long as it will be designed to cover all of the combinations of valid and invalid input as well as use of the software. Test data will be constructed to perform each of these tests and often the expected behaviour or output from each will be documented. The tests will then be carried out and the behaviour of the software solution compared to the expected result in each case.

There is some argument that the best time to compile a list of tests is during the design stage. Doing so gives those coding the solution a valuable insight into the exact parameters of both the input and output.

The final step of the testing process is correcting those errors that have been detected, after which the test data that initially triggered the incorrect result is tested again to ensure that the software solution has been fixed.

Documentation

Documentation is then written to support the variety of users of the software solution. Documentation can take a number of forms. Internal documentation is placed inside the program code and assists future programmers who wish to modify the software solution. System support documentation can be in electronic or hardcopy form. Different types of system support documentation can be produced for different groups from those using the system to those maintaining it.

4. Evaluation

The evaluation of a software solution is an important activity that will usually take place after the solution has been in full operation for a while. This time period varies, but it needs to be long enough so that users of the system are comfortable with it and hopefully not resentful of it (as can happen when new software solutions are introduced). An evaluation can (and should) contain many different elements. There are two key activities involved in this stage: evaluating the software solution and determining a strategy that will be used to find out the extent that the solution meets the required needs.

Evaluating the solution

In the design stage, a set of evaluation criteria were created that can now be drawn upon to evaluate how well the solution has met requirements, needs or opportunities. In framing the evaluation of these criteria, it is important to consider the overall efficiency and effectiveness of the solution. For that reason, criteria that have been written in a quantifiable way containing efficiency and effectiveness measures, will enable a software developer to quickly determine the extent to which a software solution has been a success or what its deficits are.

Determining a strategy

What are the best ways to find out if the software solution has met the required needs? A strategy to determine this will include a timeline for the evaluation, what data will be collected using what methods and how the data relates to the evaluation criteria set out in the design stage. Note that this activity is more complex than simply asking a series of questions of the users.

Context Questions

1. What is the purpose of the problem-solving methodology?
2. What possible consequences are there in evaluating a software solution too early?
3. What is meant by the term ‘due diligence’?
4. Is the PSM always implemented in a linear fashion? Explain what this means.
5. What is validation and in which stage does it take place?
6. When considering how to test a software solution, why is it useful to create a list of all of the tests that will be conducted?
7. What is scope and how is it different to constraints?
8. When creating a strategy for evaluating a software solution, what elements need to be included?

Applying the Concepts

- Imagine you are beginning the process of writing this text from scratch, using the PSM as the method that will guide you. Write down what you envision you would be completing at each stage and each activity within the PSM. Compare this to what others have and discuss any different approaches that become evident.

Key Skills Checklist

At the conclusion of this chapter, you should be able to address the following key skills. Mark each off as you can achieve them.

Understand the stages and activities within the PSM	☐
Explain the difference between functional and non-functional requirements	☐
Understand what it means for a solution to be effective	☐
List and describe measures of effectiveness	☐
Understand what it means for a solution to be efficient	☐
List and describe measures of efficiency	☐

Sample Examination Questions

The following sample examination questions can be attempted to test your knowledge of the content of this chapter.

Question 1
The purpose of the Problem Solving Methodology is:

- **A.** To give form to what goes on in completing a task
- **B.** To document a process within an organisation
- **C.** To create a solution to a problem
- **D.** To provide a structure by which a solution can be found to an information problem

Question 2
Which of the following lists contain measures of efficiency?

- **A.** readability, cost, attractiveness
- **B.** speed of processing, cost, level of automation
- **C.** speed of processing, usability, accuracy
- **D.** cost, productivity, timeliness

Question 3
Which of the following is a functional requirement of a solution?

- **A.** User friendly
- **B.** Robust
- **C.** Calculates the tax payable for the financial year
- **D.** Can be easily maintained

Question 4
The four stages of the Problem Solving Methodology are:

- **A.** Analysis, Design, Implementation, Feedback
- **B.** Design, Implementation, Feedback, Packaging
- **C.** Analysis, Development, Evaluation, Construction
- **D.** Analysis, Design, Development, Evaluation

Question 5
With reference to the stages of the Problem Solving Methodology, the stage that involves writing evaluation criteria is called the _____ stage.

1 mark

Question 6
With reference to the stages of the Problem Solving Methodology, validation takes place within the _____ stage.

1 mark

Question 7
What is the difference between efficiency and effectiveness?

2 marks

Question 8
Paul is beginning the process of developing an App that will be used for podcasting. He starts by gathering some data from both the presenters and listeners about what they would like to see included. For each of these groups, list one functional requirement that may be put forward as well as one non-functional requirement.

Group	Requirements
Presenters	Functional: Non-functional:
Listeners	Functional: Non-functional:

4 marks

Sample Examination Answers

Question 1
Answer: D

Question 2
Answer: B

Efficiency vs effectiveness. Straight from the glossary definition.

Question 3
Answer: C

A functional requirement is one that the solution can do.

Question 4
Answer: D

Question 5
Answer: Design

Easy mark here – definition directly from the PSM.

Question 6
Answer: Development

Question 7
Efficiency is a measure of how well a task is done while effectiveness is a measure of how correctly it is done.

Efficiency and effectiveness are mentioned frequently in the study design and feature in the exam. Often it will not be in a question as simple as this one. It is important to use these terms in your answers but at the same time ensure that you use them in context and explain why. For example, stating that a new system is efficient will not earn you any marks, but stating that a new system will increase efficiency by reducing processing time, will.

Question 8

Group	Requirements
Presenters	Functional: Sound quality is high. Non-functional: Easy for listeners to find their show and listen to it
Listeners	Functional: Easy to navigate and select shows for their play-list Non-functional: Reliable / doesn't crash or lose where they are up to in their play-list

Functional and non-functional requirements are a common exam question and it is important to not only know the difference between them, but to be able to list things that come under each category.

Chapter 2
The Structure of a programming language

The chapter covers Unit 3: Area of Study 1 key knowledge:

- *Data and information*
  - *KK1.1 Characteristics of data types*
- *Approaches to problem solving*
  - *KK1.5 Methods of representing designs, including data dictionaries, mock-ups, object descriptions and pseudo-code*
  - *KK1.7 A programming language as a method of developing working modules that meet specific needs*
  - *KK1.8 Naming conventions of solution elements*
  - *KK1.9 Processing features of a programming language, including classes, control structures, functions, instructions and methods*
  - *KK1.14 Purposes and characteristics of internal documentation, including meaningful comments and syntax.*

Key terms: binary, kilobyte, data types, boolean, floating point, integer, string, date, control structure, sequence, selection, repetition (iteration), subroutine, function, method, event, class, internal documentation, naming convention, mock-ups, data dictionary, object descriptions, algorithm, top-down design.

Study roadmap

	Unit 3		Unit 4	
Ch	Area of study 1 – Programming	Area of study 2 – Analysis and Design	Area of study 1 - Development and Evaluation	Area of study 2 - Cybersecurity: software security
1	✓			
2	✓			
3				
4				
5				
6				
7				
8				
9				
10				

What is binary?

Within any computer, the basis of data representation is essentially the presence of particular voltages on a series of wires. If there is a voltage on a single wire, then it is said to be carrying the value '1'. If there is no voltage, then it is said to be '0'. This is the basis of binary and all signals within a computer are represented in this way.

Within the first computers that were built, these signals were moved around the inside of the computer (on the motherboard) through the use of parallel wires known as a bus. These first buses consisted of eight wires. As binary is a base 2 number, the total number of combinations that can be represented using eight wires (that is from '00000000' all the way up to '11111111') is 256. As the bus on a computer's motherboard serves the purpose of moving data from one section of the motherboard to another and a great deal of what is moved around inside a computer is

actually text, a code was needed to represent this.

ASCII (the American Standard Code for Information Interchange) was created in 1963 to serve as one of the standard systems for representing text in computer systems. The ASCII table translates 7-bit binary codes to the letters of the keyboard and other special characters and is still used today (although its form has changed).

A single binary digit is known as a 'bit' and 8-bits are known as a 'byte'.

Larger chunks of binary

Soon the amount of binary numbers that we were storing grew larger, as did the width of the computer bus on the motherboard. By increasing the size of the bus, computers were able to move more data around more quickly. Larger units were needed to represent these greater quantities.

A kilobyte represents 1000 bytes ('kilo' means 1000). Technically speaking, as binary is a base 2 number, a kilobyte is not exactly 1000 bytes, but is equal to 1024 bytes (which is 2 to the power of 10). 'Kilo' was adopted as the prefix as the number was so close to '1000' that it was the most convenient measure. Each increase of scale (by 1000) has seen a different unit defined. Figure 3 summarises the ones that are used commonly today (or soon will be).

Characteristics of data types

Data types: The forms that an item of data can take, including binary (as represented in images and sound), Boolean, character and numeric, characterised by the kind of operations that can be performed on it. Depending on the software being used, these data types can be divided into more specific types, for example integer and floating point, which are numeric types. More sophisticated types can be derived from them, for example a string of characters or a date type, and their names may vary, such as text data type versus string data type.

VCAA VCE IT Study Design 2020-2023 Glossary

An understanding of the data types that can be utilised in a programming language is paramount to the creation of an efficient software solution. Many would argue that as memory capacities and CPU clock speeds have increased, the need for the careful selection of data types has become less of an issue. However, the choice of variables used in a software solution has direct implications for storage sizes, processing times and accuracy. The larger the program being developed, the more influence these choices have on the efficiency (running time and memory used) of the finished solution.

Figure 4 on the next page lists some of the different variable types and their typical

Units of storage

Name	Abbreviation	Size	Power of 2
Byte	None	8 bits	8
Kilobyte	KB	1,024 bytes	10
Megabyte	MB	1,048,576 bytes	20
Gigabyte	GB	1,073,741,824 bytes	30
Terabyte	TB	1,099,511,627,776 bytes	40
Petabyte	PB	1,125,899,906,842,624 bytes	50
Exabyte	EB	1,152,921,504,606,846,976 bytes	60

Figure 3: Units of storage

ranges. Different names can be used for these types in each language and their storage size and value range can vary. There are also many other data types that are available, but our focus in this chapter is on the standard types listed in this table.

Some of the ranges in Figure 4 may appear quite strange. They are however, not arbitrary values, but are determined by calculating the maximum number that can be stored using that many binary digits.

The structure of a programming language

Programming languages differ from each other in syntax and style, but there are elements that are common to many of them. Features and elements within programming languages have changed over the years with the advent of new generations of languages (referred to as 'GLs', for example, 3GL or 4GL). Many of the programming languages that are in use today are 3 and 4GLs, but 5GLs are in development.

Languages are often classified as being either high or low level languages. The distinction between these is based on the amount of translation that is needed to take the language from its entered form to binary code (known as compiling). The more translation required, the "higher" the level of language. It used to be the case that if a software developer wanted to achieve higher levels of efficiency, they would code in a lower level language so that they could better control what was happening at machine level. This is still true, but many of the modern programming languages strike a good balance between ease of use and allowing the user to access machine level control if they wish.

Instructions and syntax

Each programming language has a syntax that needs to be followed for the structure of the code as well as the syntax of the commands. Programmers will often make use of language reference material (which is a form of documentation) which will describe the syntax of all of the available commands.

Variable types and their typical ranges

Variable Type	Storage Size	Value Range
Boolean	1 byte	True or False
Character (or Char)	2 bytes	0 – 65,535
Floating Point (or Decimal or Real)	12 bytes	+/- 79,228,162,514,264,337,593,543,950,335 with no decimal places +/- 7.9228162514264337593543950335 with 28 decimal places (maximum) Smallest non zero number: +/- 0.0000000000000000000000000001 or (+/-1E-28)
Integer	4 bytes	-2,147,483,648 – 2,147,483,647
String	Depends on the string length.	Depends on the string length.

Figure 4: Variable types and their typical ranges

Control structures

There are three main control structures that every programming language contains. These are sequence, selection and repetition (or iteration) structures. These control structures are the core of any language and, without them, the language would not be able to function effectively. A key part of learning to use any programming language is learning how to code these control structures. Examples of how these are structured will be shown shortly.

Subroutines (or procedures)

The simplest sort of structure within a programming language is often a subroutine, also known as a procedure, function or method. Subroutines have been part of programming languages since the third generation. A subroutine is a contained section of code that can be called from within any other part of the program (even from within the subroutine itself). Subroutines primarily prevent programs becoming large with big sections of repeated code. Code that is useful and is being used frequently can be placed into a subroutine and called as needed.

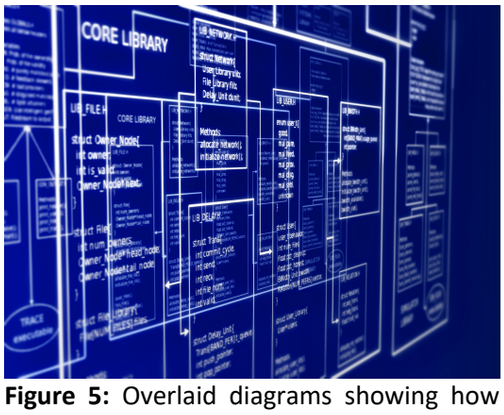


Figure 5: Overlaid diagrams showing how libraries, subroutines and methods are often represented

To aid in flexibility, subroutines can also accept parameters to vary the way they operate. These parameters can be passed by value or by reference, which basically means the difference between passing a copy of the data to the subroutine, and passing a pointer to the variables themselves (which would then mean that the subroutine could alter them).

Subroutines are very useful in dividing a project up amongst a programming team. As they can be written to accept parameters and change the values of these parameters by reference, they can be coded almost like independent programs. Subroutines that are very useful can also be placed into program libraries and used again and again across a variety of projects.

Functions

A function is very similar to a subroutine and in many ways can be considered a type of subroutine. A function performs a task that is frequent or useful to the programmer and often is used in a way that passes data back to the calling code. Common examples of these are mathematical functions such as SIN(X) and COS(X). Functions like these ones accept a parameter (in this case 'X') and pass data back via the name of the function.

Both procedures and functions can be considered to be subroutines and their syntax and use is defined by the language in which they are being used. A common distinction between procedures and functions has been that procedures do not return data and functions do. Although this is commonly true, some languages do not define procedures and functions in this way and indeed, even languages that do, allow them to be used in ways that are contrary to this.

Methods and events

Object oriented (OO) programming languages (4GL) work in a very non-sequential way, as opposed to many of the 3GL languages. Although languages such as Visual Basic.Net have their origins in the third generation, they work in a fundamentally different way, triggering instead on events and utilising methods within the code.

Methods and events are similar in ways and in this sense hard to separate. Events usually refer to those things that happen while the program is running that are triggered by a user's interaction or by another object. When an event is triggered, the program executes the code that is contained under the heading of the object and event in question. In this sense, events are usually the equivalent to subroutines in 3GLs. For example, a button object may contain code in several different event procedures, one of which could be a 'click' event. When the user of the program clicks on the button, the code inside the click event procedure would be executed.

Methods are commands that can be used to interact directly with objects to change their behaviour or have them perform a particular function. Every object in an OO programming language will have certain behaviours associated with it that can be accessed via a method. As the type of object can be quite varied, these methods are usually quite specific to the object type in question. The important aspect of this to understand is that methods belong to objects. All objects have specific methods that belong to them, and in some cases, these methods may be unique.

Classes

Bringing these concepts together, is the object known as a class. A class is a definition of an object that has a number of methods and events associated with it. It can be duplicated and used independently of the other instances of itself. Just as a variable can be declared to be of a particular type, an object can be created that is of a particular class and named in a unique fashion that allows it to be referenced on its own.

To use an analogy to explain this, imagine you have at your disposal a small robot dog. You are able to interact with the dog in a finite number of ways. You press a button on the dog's back a number of times to give it commands, pat the dog, 'give' the dog a toy bone or pick up the dog. All of these interactions are recognised by the robot dog and can be considered to be 'events'. The robot dog is also able to perform a number of specific actions. The dog can walk, run, heel, beg and 'bark'. All of these actions can be considered to be the 'methods' that belong to the dog (that is, what the robot dog can do). All of these things can be considered to form the class of robot dog.

If another robot dog were to be created using this class of robot dog, it too would be able to be interacted with in the same ways and would be able to perform the same actions. It would however act independently from the first robot dog and any other robot dogs that were obtained and placed in the same environment. Most 3GL or 4GLs have the ability to define classes and doing so is a powerful way of building sophisticated software solutions drawing on previous code and structures rather than writing everything from scratch.

The role of internal documentation

As well as writing documentation to enable the users to use the program effectively, it is equally important to write internal documentation in the program's code. Although there may not be many circumstances in which programmers other

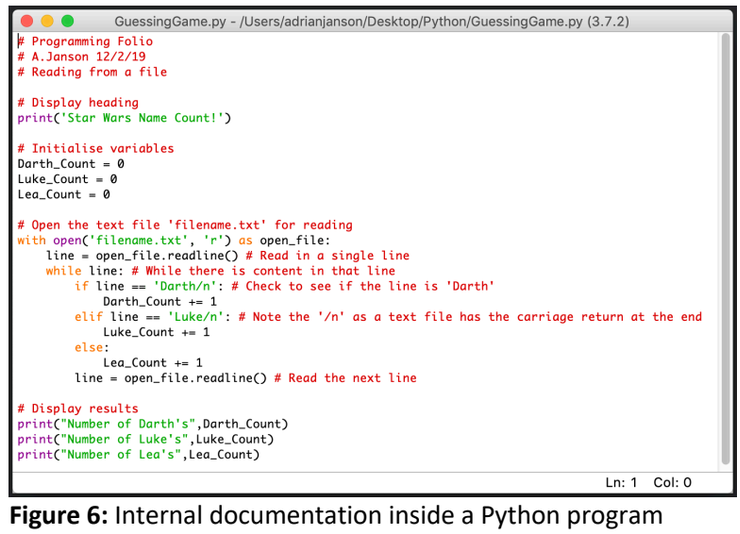


Figure 6: Internal documentation inside a Python program

than those that coded a program initially will want to examine the code, internal documentation is a vital element in the software development process. If the program has been created by a programmer while employed by a company, the company owns the program and may wish to make changes long after the original programmer has left the company. Similarly, some aspects of a program can be quite complex, and even the original programmer may have trouble interpreting uncommented code if they need to make changes to it in the future.

Internal documentation should describe the function of key variables and procedures and should also include an explanation of the naming and coding conventions that have been used. It is also usual to include any references to code that has been sourced from places other than the author or the company itself as well as include revision information describing who has contributed to the software development and when.

Internal documentation does not have any effect on the efficiency of a software solution as the compilation process ignores all comment lines and does not convert them to machine language.

Good internal documentation practices

Header comments

At the beginning of a module or software solution, a number of lines can be used to state the name (or names) of those that have written the code, the date that the code was last updated, the version number of the code and a general description of its purpose. Doing this gives another developer some initial information about the code and may aid their understanding.

Use of white space

Blank lines should be used to separate groups of statements that have a different purpose. The use of blank lines makes the code look less dense and aids readability.

Communicate the intended purpose

Make sure that when you are writing internal documentation, you are explaining what the code should be doing as opposed to what function individual statements have. These two things may seem to be the same, but to another developer trying to understand what the code is doing, it will be easier for them if they know what the purpose of the statements are as opposed to simply what they do.

Write your comments for someone else

In many cases, internal documentation will be used by the software developers that wrote the original code. However, there will be many times when developers are called upon to write modifications to existing code within an organisation. With this in mind, internal documentation should always be written clearly and professionally.

Naming conventions for solution elements

Whether writing an algorithm or writing code, it is very important to decide on a naming convention that will be used for all the variables and subroutines within the program. This convention should then be applied consistently throughout and documented in some fashion, possibly through internal documentation and certainly in some hard copy format.

The use of a naming convention is preferred for a number of reasons. In the long run, a naming convention aids the programmer by reminding them of the function of variables and subroutines that they may not have used or accessed for a while in the development of the software. In addition to this, naming conventions aid those who are reading the program with a view to understanding or modifying it later on.

'Hungarian notation'

An example of a naming convention is the 'Hungarian notation' convention. 'Hungarian notation' is a popular naming convention for programmers in all programming languages, particularly 3GL. It has a number of rules related to the naming of subroutines, variables and objects. When a subroutine, variable or object is placed into a program, it should be named in the following way: the first few letters (all lower case) should indicate the type of the particular subroutine, variable or object. It doesn't really matter what letters are used, but it is important to be consistent. A button could use the letters 'btn' and a label could use the letters 'lbl'; similarly, an integer may begin with the single letter 'i', but an integer tracking the size of a list or array may begin with the characters 'sz'. After these type-distinguishing letters, a word or number of words should be added that describe the purpose of the subroutine, variable or object. They should be written in 'camel case' – where each word begins with a capital letter.

For example, a button that is being used to calculate a tax rate could be named 'btnTaxRate'. A text box that is to be used for the entry of a person's name could be called 'txtUserName'.

The origins of Hungarian notation

In the early days of programming language development, the Chief Architect at Microsoft, Dr. Charles Simonyi, introduced an identifier naming convention that added a prefix to the identifier name in an effort to label the function or type of the identifier. The naming convention proved to be very popular as it made the code easier to read by others and many of Dr. Simonyi's colleagues began using the convention for themselves. As it had no formal definition, and in part because Dr. Simonyi was Hungarian, it became known as the 'Hungarian notation'.

Methods of representing designs

When considering the process of moving from the definition of a problem and its scope, to the design of potential solutions, a number of tools will be used.

Annotated diagrams or mock-ups

By creating annotated diagrams (or mock-ups) that model the layout of the proposed software solution, a software developer can easily convey the user interface and the way that the solution will operate. A client can examine the annotated diagrams and give feedback based on the way the designs address the functional requirements of the solution. They can also give feedback on the way the user interface looks in terms of some of the attributes of user interfaces discussed later in this chapter.

Data dictionaries

A data dictionary is a table that lists and describes all of the variables that are (or will) be used in a program or suite of programs. It essentially lists the meta-data for the variables that a software solution will be using.

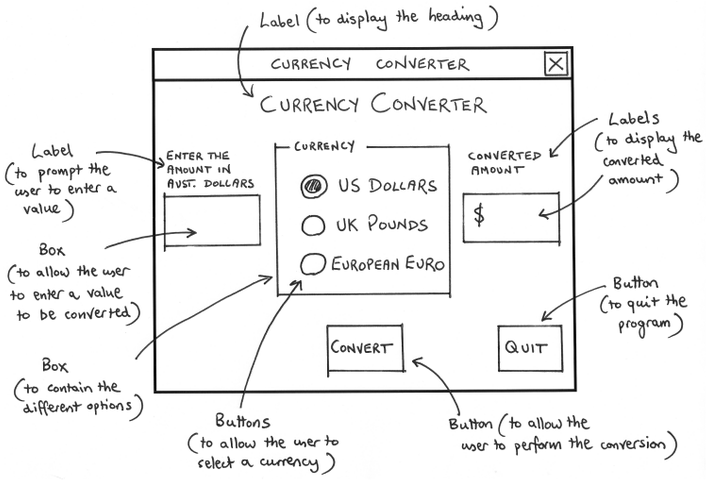


Figure 7: Annotated diagram for a currency converter program

Usually, it lists the names of the variables, their type, size (in characters), scope and a description of their function. Data dictionaries can be created that contain more information than this depending on personal preference. An example is shown in Figure 8.

Variable Name	Type	Size	Scope	Description
ID_Number	Integer	6	Global	ID number of the product
strName	String	10	Global	Name of the product
strType	String	10	Global	The type or category of the product
boolGST-Exception	Boolean	1	Global	Is the product GST free or not
… etc.				

Figure 8: An example of a data dictionary

Object descriptions

Object descriptions assist with the planning of the structure and content of objects that are to be incorporated into the design of a software solution. There are many different ways of representing object descriptions. The programming language being used will have an effect on the content of an object description. Object descriptions are similar in style to data dictionaries and can contain information such as the methods and events associated with an object. Figure 9 is a typical example of an object description. Note that an explanation of methods and events can be found in the next chapter.

Both data dictionaries and object descriptions can be represented in a wide variety of ways. There is not one standard for representing these types of design tool, but the important thing is that you understand the purpose and function of both.

Algorithms

Once the requirements of a program are known, an algorithm can be created. An algorithm is a set of ordered steps or instructions that accomplish some task or goal. A program will be written in a specific computer language but an algorithm is not in any particular language. It defines, in English the basic steps needed to solve the problem - similar to a recipe. Just like a recipe, algorithms also have specific symbols and formatting dependant on the type of algorithm being used.

Programmers can sometimes be resistant to writing algorithms, but doing so has some significant benefits. The main benefit is that problems can be avoided early - especially those that are logic based. They also allow the scope of a task to be determined, which can be helpful when a team of programmers are setting themselves up to begin a large programming task.

Object Name: btnCalculate

Name	Type	Description
btnCalculate_Click	Event	When the button is clicked, it will calculate the result of the current transaction and display the result on the screen.
btnCalculate_DblClick	Event	When the button is double clicked, it will lock the calculate function so that no other buttons can be clicked.
btnCalculate.Enabled	Method	This method will be used to disable or re-enable the button while the program is running. It can be set to ‘true’ or ‘false’.
btnCalculate.Visible	Method	This method will be used to make the button visible or invisible based on the user’s access level. It can be set to ‘true’ or ‘false’.
btnCalculate.Focus	Method	This method can be used to place the cursor on the button so the user can only select this. It will be used at appropriate times to guide the user through the ordering process.

Figure 9: An example of an object description

Top-down design

Just as writing a program from scratch is a daunting task, writing an algorithm from scratch is equally daunting. Software developers often use a method for breaking the problem down in small parts, known as "top-down design".

The first step in writing any algorithm is defining what is to be achieved. That is, what is the goal of the program? This may be outputting the result of a calculation, displaying a list of various items or communicating across a network.

The next step is determining what information is required in order to achieve this goal. What information will need to be gathered or input? What processing needs to occur?

When considering these things, a rough order of tasks that need to be performed should be constructed. These tasks may be quite large at this stage of the process, but that's the way that top-down design works. Each task can now be broken down into smaller, more manageable parts. If these smaller parts are still too large, they may be further broken down until the products of these divisions are very simple programming tasks to achieve.

Algorithmic representations

There are three different control structures that an algorithm can employ. These are sequence, selection and repetition (or iteration). By using these three structures in any combination, any logic problem can be solved. A small definition of each follows below:

Sequence

Tasks that are performed in sequence lead from one to the other in that order. Any number of tasks may be executed in sequence. Each task must be performed once – there is no possibility of skipping a task or branching off to perform a different task.

Selection

In a selection structure, a condition is tested (or a question is asked), and depending on the answer, one of a number of courses of action (usually two) is taken. After this has been done, the program moves on to the next task in sequence. A selection structure is sometimes referred to as an '**if-then-else'** statement because of the way in which it instructs the program.

Repetition (or iteration)

In a repetition structure, a number of tasks are performed a set number of times or until a condition is met. Three types of repetition are available: fixed, test at end and test at beginning. Examples of each of these structures can be found in the following sections.

A common method of representing an algorithm is using a type of algorithmic language known as pseudocode. Pseudocode was originally designed to be easy to write and understand and it is preferred by many software developers for these reasons.

Pseudocode

Pseudocode algorithms are written in plain English. As with any language, there are rules that need to be followed, although there are not many of these to remember.

The following conventions should be used when writing pseudocode algorithms:

1	Start … Stop Or Begin … End	These should be used to indicate the beginning and end of any program or subprogram. It does not matter which pair is used, as long as there is consistency throughout the algorithm.
2	Action 1 Action 2	Sequence is represented by writing statements underneath each other as shown.
3	If Condition Then Action 1 Action 2 End If	Selection is represented like this. If the condition is true, then the statement(s) following the 'then' are carried out.

4	If Condition
	Then
	Action 1
	Action 2
	Else
	Action 3
	Action 4
	End If
		Expanded version of the above convention. If the condition is true, then the statement(s) following the 'then' are carried out. Otherwise, if the condition is false, a set of different actions are taken.

5	Case of *variable* *value1*: Action 1 *value2* to *value3*: Action 2 *value4*+: Action 4 End Case	A 'Case' statement can be used to simplify conditions where there are many different values that are being checked. An action can be performed if the variable is a specific value, within a range or greater than a certain value.
6	Count ← *number*	Assigning values to variables is done with an arrow, which serves to show the flow of data. Note that conditions should still use the '=' sign when two objects are compared to each other.
7	For Count ← *first* to *last* Action 1 Action 2 Next Count	Repetition can be performed in a variety of ways, and pseudocode has a different way of representing each. This is a fixed loop that repeats a set number of times.
8	Repeat Action 1 Action 2 Until Condition	This is used to continually repeat a loop until some condition becomes true. This is an example of the 'test at end' type of repetition discussed earlier.

9	While Condition Do
	Action 1
	Action 2
	End While
		This loop is used when a condition must be true before starting the loop. This is an example of the 'test at beginning' type of repetition also discussed earlier.

10	Process, Display, Increment Add, Multiply, Divide, Subtract Calculate, Sum Input, Output <>, >, <, <=, >=	These are some examples of the more commonly used pseudocode terms. It is not important to use all of these, but consistency is important. '<>' is used to indicate 'not equal to'.

Figure 10: Pseudocode conventions

Note that the main difference between the 'test at end' and 'test at beginning' repetition structures, is that if the condition is such that the loop will end, the 'test at end' structure will still execute what is inside it once while the 'test at beginning' will not execute what is inside it at all.

The example in Figure 11 on the next page, is a typical algorithm which starts by setting a variable 'Pass' to 'unlucky'. The user is then prompted to enter a password and this is placed into the variable 'Password'. A 'test-at-beginning' loop compares the password that has been entered to the one held in the variable 'Pass'. If they are not the same, the

user is prompted to re-enter the password and it is checked again in a continuous loop. If the user correctly enters the password, the loop finishes and an 'Access granted' message is displayed on the screen.

Pseudocode example 1

```
Start
        Pass ← "unlucky"
        Display "Enter the password"
        Input Password
        While Password <> Pass
                Display "Wrong password – try again"
                Input Password
        End While
        Display "Access granted"
Stop
```

Figure 11: Pseudocode example for a simple password program

You will see in the example in Figure 12, that the actions are described in English and the structure of the problem is well laid out and easy to read. Some of the structures described above have been used and the algorithm has been indented where appropriate. This example one shows how a pseudocode algorithm can be used to describe a real-life task – in this case, washing the dishes.

Pseudo code example 2

```
Start
{Washing the dishes}
        Put plug in sink
        Fill sink with hot water
        Add detergent
        While dishes left to wash Do
                Take dish from bench
                Put dish in water
                Repeat
                        If dish is a pot then
                                Scrub pot with steel pad
                        Else
                                Wipe dish with sponge
                        End If
                Until dish is clean
                Put dish in drying rack
        End While
Stop
```

Figure 12: Pseudocode example for washing the dishes

Context Questions

1. Why is a kilobyte equal to 1024 bytes and not 1000 bytes?
2. What is the difference between a high level and a low level language?
3. Why is it important to understand the syntax of a programming language?
4. What benefits does using subroutines have over not using them at all?
5. What is the difference between a subroutine and a function?
6. What is the difference between a method and an event?
7. Explain why the inclusion of internal documentation does not have an adverse effect on the efficiency of a program.
8. Give reasons why the use of a naming convention is beneficial.
9. What advantages are there in using coding conventions for the naming of program elements?
10. In what ways can a data dictionary assist a software developer in the design process?
11. Describe the process known as 'top-down design'.
12. List the three different control structures that an algorithm can employ.
13. What does a 'Case' statement do that makes it useful to a software developer?
14. What implications does the choice of variable type have on the efficiency of a software solution?

Applying the Concepts

- Write an algorithm to describe a household chore that you currently perform.

Key Skills Checklist

At the conclusion of this chapter, you should be able to address the following key skills. Mark each off as you can achieve them.

Explain how data is represented inside the computer	☐
List and explain the differences between the main data types	☐
Create pseudo-code examples of the sequence, selection and repetition control structures	☐
Explain the difference between subroutines, functions, events and methods	☐
Explain what classes consist of and their benefits	☐
Discuss the benefits of internal documentation and naming conventions	☐
Create an annotated mock up to represent the design of an interface	☐
Understand how data dictionaries and object descriptions are used	☐
Write simple pseudo-code algorithms	☐

Sample Examination Questions

The following sample examination questions can be attempted to test your knowledge of the content of this chapter.

Question 1
The process of breaking a problem down into smaller parts is known as:

- **A.** an algorithm
- **B.** top down design
- **C.** bottom up design
- **D.** sideways refinement

Question 2
A character data type stores data that is:

- **A.** A single ASCII character
- **B.** A prime number
- **C.** A positive integer
- **D.** True or false

Question 3
Which of the following is **not** a benefit of writing an algorithm?

- **A.** Find errors in logic before coding
- **B.** Test features of the user interface
- **C.** Makes the processing more efficient
- **D.** Makes it easy to divide the project amongst members of a team

Question 4
Which of the following is **not** a structure available in an algorithm?

- **A.** Manipulation
- **B.** Sequence
- **C.** Iteration
- **D.** Selection

Question 5
Internal documentation is:

- **A.** A user guide which is included inside the code of a software solution
- **B.** Documentation that is only intended for those internal to the organisation
- **C.** Documentation intended for those that might make changes to the code
- **D.** Documentation that is opened by typing an administrator password

The following information is required for Questions 6 and 7.

btnProcess

Name	Type	Description
btnProcess_Click	Event	Calculate the final result
btnProcess_DblClick	Event	Lock the button
btnProcess.Enabled	Method	Lock or enable the button

Question 6
The tool displayed above can best be described as a:

- **A.** Context Table
- **B.** Data Flow Diagram
- **C.** Object Description
- **D.** Data Dictionary

Question 7
The person coding this object wants to add a tool tip that will appear when the mouse hovers over the object. This would be classified as:

- **A.** A method
- **B.** A variable
- **C.** A boolean value
- **D.** An event

The following information is required for Questions 8, 9 and 10.

Variable Name	Type	Size (bytes)
ID	Integer	1
Title	String	10
Classification	String	10
Borrowed	Boolean	1

Question 8
The tool displayed above can best be described as a:

- **A.** Context Table
- **B.** Data Flow Diagram
- **C.** Variable List
- **D.** Data Dictionary

Question 9
The maximum number of unique 'ID' numbers that could be stored would be:

- **A.** 16
- **B.** 256
- **C.** 255
- **D.** 2056

Question 10
A Boolean data type stores data that is:

- **A.** Encrypted
- **B.** A prime number
- **C.** A positive integer
- **D.** True or false

Question 11
It is always good programming practice to use a naming convention.

- **a.** Describe one naming convention that can be used when naming variables.

_____

_____

1 mark

b. Give **two** advantages of using a naming convention.

Advantage 1: _____

_____

Advantage 2: _____

_____

2 marks

c. Explain how not using a naming convention can lead to increased development costs in the future.

_____

_____

1 mark

Question 12

a. What is internal documentation?

_____

1 mark

b. Describe a situation in which not having any internal documentation would be a problem.

_____

_____

1 mark

c. Martin and Julee are having a debate about internal documentation. Martin believes that every line of code should be accompanied by at least one line of internal documentation. Julee argues that a few lines of documentation per event procedure is enough.

Discuss the pros and cons of each point of view.

_____

_____

_____

_____

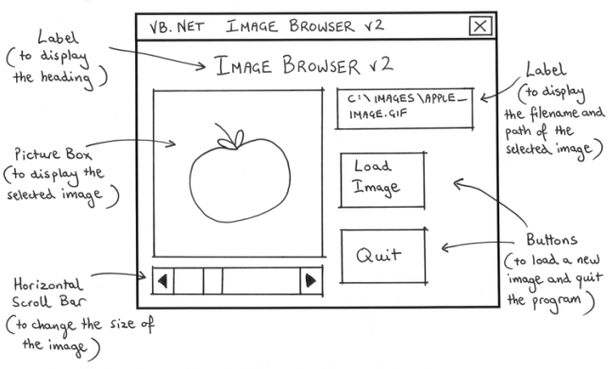
_____

4 marks

Question 13
A variable type that can only have two values: 'true' or 'false' is called _____.

1 mark

Question 14
Consider the diagram shown below:



a. What is the name for a diagram of this type?

_____

1 mark

b. Who has this diagram been produced for?

_____

1 mark

c. At what stage of the Problem Solving Methodology would a diagram such as this be made?

_____

1 mark

d. List two benefits of producing a diagram of this type (as opposed to not producing one at all).

Benefit 1: _____

Benefit 2: _____

2 marks

Question 15
What is the difference between a method and a function?

_____

_____

2 marks

Sample Examination Answers

Question 1
Answer: B

Question 2
Answer: A

Though programming languages used across schools vary, the standard data types listed in the study design are the ones that you need to be familiar with.

Question 3
Answer: B

The design of a user interface, while it also takes place in the design stage, is a different process to writing an algorithm.

Question 4
Answer: A

Manipulation is something that gets done in an algorithm, but rather this is spelled out using sequence, selection and iteration.

Question 5
Answer: C

Question 6
Answer: C

Question 7
Answer: D

This question is aimed at the difference between methods and events. Events are (generally) triggered when objects are interacted with, though events can also be triggered by other objects or the object itself. A method is contained within the definition of the object and can be used to change its function or appearance.

Question 8
Answer: D

Question 9
Answer: B

1 byte has a maximum number of 255 – plus 0, makes 256 different numbers.
Question 10
Answer: D

Question 11

- **a.** Hungarian convention.
- **b.** i – Makes it easier to debug the program, ii – Makes it easier to see the function of variables (their type, scope and purpose)
- **c.** Anyone doing further development on the program will need to decipher what the variables do and this will take time – leading to an increase in costs.

Question 12

- **a.** Internal documentation relates to comments (documentation) placed within the code to make it easier for programmers to understand and modify the code at a later date.
- **b.** Not having any internal documentation makes modifying the code later on difficult – especially if it is being done by someone other than the person that wrote the code.
- **c.** Martin: documenting each line of code will be very time consuming. It will, however, mean that the code is very easy to understand at a later date.
  Julee: documenting each event procedure with a few lines will be much more practical and less time consuming. However, there will be event procedures that are long and complex and if Julee limits her comments to just a few lines, this may not be enough to fully explain the function of the code.

When a question uses the stem 'discuss', this means that you need to discuss the pros and cons of two alternatives. This is also reflected in the marks for the question which are set at 4 – meaning that the examiners will be expecting you to make 4 points.

Question 13
Answer: Boolean

Definition of a variable type.

Question 14

- **a.** annotated diagram or a mock up
- **b.** software developer
- **c.** design
- **d.** 1: have a clear idea of what the solution will look like and how it will be laid out
  2: can be used to show the client and gain feedback on the design

Question 15
A function is a subroutine that returns a value. A method is a subroutine that is associated with a specific object.

Chapter 3
Data structures and data manipulation

The chapter covers Unit 3: Area of Study 1 key knowledge:

- *Data and information*
    - *KK1.2 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)*
- *Approaches to problem solving*
    - *KK1.6 formatting and structural characteristics of files, including delimited (CSV), plain text (TXT) and XML file formats*
    - *KK1.10 Algorithms for sorting, including selection sort and quick sort*
    - *KK1.11 Algorithms for binary and linear searching*
    - *KK1.12 Validation techniques, including existence checking, range checking and type checking*
    - *KK1.13 Techniques for checking that modules meet design specifications, including trace tables and construction of test data.*

Key terms: data structure, one-dimensional array, record, file, associative array, hash table, linear search, binary search, selection sort, quick sort, files, CSV, TXT, XML, validation, existence check, range check, type check, testing, syntax error, logic error, run-time error, debug, trace table, testing table, test data.

Study roadmap

	Unit 3		Unit 4	
Ch	Area of study 1 – Programming	Area of study 2 – Analysis and Design	Area of study 1 - Development and Evaluation	Area of study 2 - Cybersecurity: software security
1	✓			
2	✓			
3	✓			
4				
5				
6				
7				
8				
9				
10				

Types of data structures

A data structure is a way of storing or organising data so that it is easier to access or more efficient to use. Let's first discuss the simple data structures that can be used, and later we will examine some of the more complex ones.

One-dimensional arrays

The implementation of an array can vary among programming languages. Stated simply, a one dimensional (or 1D) array is a data structure in which variables are grouped together under the same name and accessed via an 'index'. The 'index' is most commonly an integer value which starts at the value '0'. Although an array typically contains only one

data type, it can consist of multiple data types (as is possible in some languages).

For example, let's say that we had four numbers that needed to be stored. We could store the four numbers using integer variables called 'Number1', 'Number2', etc. In memory, this would look like the diagram shown in Figure 13.

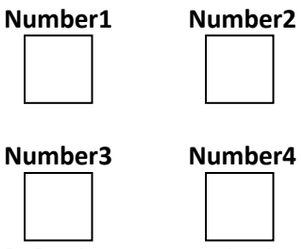


Figure 13: Storing numbers in single variables versus an array

There is nothing wrong with declaring these four variables and using these in a program. The problem with this sort of method arises in the way in which the variables will be used. If the programmer wishes to ask the user to enter a number into each of these variables and then display a total, it would probably be done as shown in Figure 14 below.

Adding four numbers together

```
Begin
        Total ← 0
        Input Number1
        Input Number2
        Input Number3
        Input Number4
        Total ← Number1 + Number2 +
        Number3 + Number4
        Display Total
End
```

Figure 14: Adding four numbers together

A 1D array would group all of the number variables together in a way that would allow them to be accessed via an 'index' which would point to the one we wanted. This could be represented like the diagram shown in Figure 15 below.

The declaration of an array of integers called 'Number' to hold 4 items would look like this:

Declare Number(4) of integer

If we were to set the value of:

Number(0) ← 20

Number

Index	0	1	2	3
Value	**20**			

If we were to execute the lines shown below:

Number(1) ← 72
Number(2) ← 37
Number(3) ← 30

Then the array would look like this:

Number

Index	0	1	2	3
Value	20	**72**	**37**	**30**

Figure 15: Storing numbers in single variables versus an array

Note that the index of an array always begins at '0'. For example, if an array is declared with 100 elements, they will be numbered from '0' – '99' (which is 100 elements in total). The reason for this is because the index is used as an 'offset' to reference the memory locations the array is using that are stored sequentially in memory. That is, the first element of the array has an index of '0' because it is located at the memory location pointed to by the variable name of the array. The second element of the array has an index of '1' as it is one memory location in advance of the first one (and so on).

The same program to input the four numbers and then calculate and display the total could look like Figure 16 on the next page.

Adding four numbers using an array

```
Begin
        Total ← 0
        For Loop ← 0 to 3
                Input Number(Loop)
                Total ← Total + Number(Loop)
        Next Loop
        Display Total
End
```

Figure 16: Adding four numbers using an array

At this point in the comparison, you may be thinking that the non-array version essentially consists of seven lines of code while the 1D array version consists of six lines of code to do the same thing. This is not a huge improvement – and you could argue that the 1D version is more complicated to code. However, this is not a good example to demonstrate the real strength of 1D arrays. If we were dealing instead with 500 numbers, the non-array version of the program would need to ask the user to input each of the 500 numbers (using 500 separate lines of code). That would increase the size of the program to 503 lines, and would mean implementing a new version of the program if the program had already been released. The 1D array version would only need some small changes to the size of the loop and would remain a program of only six lines.

Records

A record is a structure that can be used to group together variables for a particular purpose. Records are similar to arrays except that where an array usually contains elements all of the same type, the variables within a record may be of different types and sizes. Indexing the elements of a record is often done via an identifier which is declared at the same time as the record.

For example, it might be necessary to store contact information for customers, like their name, address, suburb and telephone number. You could certainly store this information in a number of arrays (using the index of the array to indicate which customer's details you were accessing). Some difficulties arise when data is represented like this. A problem is that each separate array would not make a lot of sense on its own. Examining the sixth phone number would not tell you whose phone number it was. You would need to also access the sixth element of the name or surname arrays to discover this. The main difficulty with such an arrangement comes when the information needs to be sorted in some way. Sorting a number of parallel arrays (and keeping them in sync with each other) is a difficult task. A record designed to store customer information would group all of the different variables together as shown in Figure 17.

Customer_Details

Name	Surname	Address	Suburb	State	Telephone

In an algorithm, values could be placed into the different parts of this record like this:

```
Customer_Details.Name ← "Phillip"
Customer_Details.Surname ← "Rivers"
Customer_Details.Address ← "217 Williams Rd."
Customer_Details.Suburb ← "South Yarra"
Customer_Details.State ← "Vic."
Customer_Details.Telephone ← "0414297564"
```

Figure 17: An example record

A record like this is quite useful, but as it stands, it is storing one set of information. In practice, records are placed into arrays so that large sets of information can be stored (which is effectively the same as a database table).

Using data structures to organise data

Associative arrays (or dictionaries)

An associative array is essentially a way of connecting two pieces of information together. It will not be the main repository for the data within a system, but will instead aid in the organisation of that data by associating a transaction or event to it. The way that it does this is by using the main key value used to index the data and then connecting it to another value to form a (key, value) pair. Key values do not have to be all represented in the associative array, but by definition, should be unique identifiers.

Let's say that we need to record the names of people that are staying in a hotel (by the room number). If a person is staying in a room, then we record their name against the room number, otherwise we don't record the room number in our array at all. An associative array to represent this might look like the one below. There are a few things to note about associative arrays that are illustrated by this example.

Firstly, an association consists of a binding and this is often shown by using a ':'. Some programming languages (PHP for example) actually make use of the ':' character in their implementation of associative arrays.

The second thing to note is that not all of the rooms in the hotel are represented. In fact, it is not clear how many rooms are present in the hotel.

The third thing to note is that a person can effectively book more than one hotel room under their name. The 'key' cannot be repeated but the 'value' can be.

A hash table is a type of implementation of an associative array.

Hash tables

A hash table is essentially a way of implementing an associative array in an array of a smaller size than the known set of key values. If an associative array is able to be used in an array that is the same size as the total number of key values, then using a hash table is not really required. Let me illustrate this with an example.

"Room101" : "Jameson, Elaine"
"Room105" : "Phillips, Mike"
"Room106" : "Canter, Judy"
"Room204" : "Jameson, Elaine"
"Room209" : "Le, Jenny"
"Room210" : "Teo, Manti"

Figure 18: Storing information in an associative array

The hotel room associative array would not be a good candidate for a hash table as the number of rooms in the hotel is a known quantity and is also relatively small. The overhead in storage and processing time is miniscule in a situation like this. However, let's instead imagine that we want to record the occurrence of a word in an essay (not how many times a word has appeared but just that the word has been used). The number of words in the English language is now just over 1,025,000. We could make an array of this size so that we could represent a strict key:value binding for every single word. While that would work, there are some problems with doing this. Firstly, the number of words in the English language is increasing, so our data structure would require frequent modification to stay in line with this. In addition, it is highly unlikely that an essay would use every single word in the English language. A typical essay may only have a few hundred unique words, so the use of an array of over one million elements will mean there is a lot of wasted

storage space and a processing overhead to examine all of these items.

Instead a better solution is to use a smaller array (as a hash table) and place values into it using a hash function.

Instead of using a very large array, perhaps we implement an array of size 1,300. This may seem like an arbitrary number to have chosen, but it has been chosen as it is 26 x 50 and there are 26 letters in the alphabet. If we use the first letters of the word as the hash function, we can place them into the array based on this. We could have made the array only have 26 elements, but then some elements of the array would become full of many words and the processing would become difficult.

The main aim in implementing a hash table is to have the elements within the table evenly spread out and easy to locate. While it is possible to have more than one item in an array element, doing this adds to the processing time. Ideally, in designing the hash table and the hash function, a software developer is trying to strike a good balance between efficiency and storage space.

In the example shown in the figure, there are only four 'buckets' for the building bricks. The hash function being used is clearly the colour of the brick. You can see that in each 'bucket' there are a number of bricks, so finding a brick of a certain size would not be as easy as it would be if there were instead three 'buckets' for each colour (for example).

Of course the reason for using a hash table is not really to store information, but to organise it in such a way that retrieving it is easy. Typically the way that you would use a hash table would be to put the 'key' into the hash function to find the location that it should be in, and then check that location to see if it is present.

Figure 19: Organising building bricks by colour (hash function)

Structuring input and output

Software developers need to consider how the software they are writing will handle input and output. There are many ways in which a software solution can receive input. The most common form of input may be via the keyboard and the mouse, but increasingly, input is gathered through a variety of devices and methods. Devices for input include tablets, smart phones, styluses, joysticks and game controllers. There are also a variety of methods of inputting data such as through touch interfaces and the movement of devices via accelerometers and GPS coordinates.

The quality of data is of critical importance, as once data is gathered, the value of any information that is produced will be contingent on this. In addition to this, information that is output from a software solution may in turn provide input to other software products. Therefore it is not only important to ensure that data is of the highest quality, but it is important to have a good understanding of the characteristics (or limitations) of that data.

Quality of data

There is a well known acronym in software development: 'GIGO' (or Garbage In, Garbage Out). 'GIGO' describes what happens when invalid (or nonsense) data is entered into a software solution. The code tries to interpret the data as valid data and so what is produced is an unknown. It may result in the program crashing or it may produce unexpected results (or 'garbage'). The process by which input data is filtered to ensure that it is valid and 'GIGO' does not occur is known as validation.

Validation techniques

There are a large number of techniques that can be used to validate data. It has to be said that the best technique is not to require any validation at all! If a user interface is designed so that the choices that users can make are limited to valid types and values, then validation (as a whole) becomes a lot easier. This is not always possible and so techniques must be employed to check the data that has been entered is as valid as possible.

Existence checking

One of the simplest ways to validate an item of input data is to ensure that the user has entered it. This is known as existence checking. In many programming languages, a blank input field is interpreted as being equal to '0', which may be a valid input into the program or may cause it to crash. An existence check will check to see if data has been entered. If it hasn't, a prompt will be displayed to the user to enter the data before proceeding. One way to prevent most occurrences of this problem is to insert default input values into the program and allow the user to change these values. However, a malicious user could still delete the default value and try to proceed to the next step in the program with the input box empty. For this reason, existence checks are almost always necessary.

Type checking

In the same way that the lack of input can cause a problem, input of the incorrect type can cause unexpected results. Entering data of a different type into an area of the program that is expecting something else could cause the program to crash or the 'GIGO' effect.

Often input boxes that are expecting text to be entered can receive input of any type – numbers, special characters, upper and lower case letters, etc. This might not be a serious problem, but it may compromise the information that is produced. More serious is when text is entered as an input that is meant to be a number. In some cases, the value of the number might be interpreted as zero, but most programming languages will crash when this happens.

Type checking also applies when a user enters a decimal number into an input that requires a whole number value. In a case such as this, a software developer may decide that it is fine to round the user's input to the nearest whole number, rather than require them to re-enter it.

Range checking

Range checking is the process of determining if an inputted value is within the range of acceptable values for the program. This is usually most applicable to number values, but can also be applied to text. For example, you may wish to perform a range check for valid Australian postcodes.

While the usual format of a range check may be to test to see if a value falls between an upper limit and a lower bound, a range check may also test to see if an inputted value is one of a number of acceptable values (perhaps stored in an array or similar data structure).

Existence, type and range checking – a logical sequence

It is highly possible that these three types of validation could be used sequentially. If a number were required to be within a certain range, it would first need to be checked for existence, then checked to see that it was a number type before the number was tested to see if it was within the correct range.

Testing

Testing is a vital step in any software development process. This not only includes making sure that the software solution does what it is intended to do, but ensures that it is error-free. Testing in this context becomes a serious legal and/or contractual issue. As it can be such a time consuming process, testing is sometimes done by people employed specifically for this task, but this is a luxury not

available to all companies as it incurs extra cost.

Automated testing can help to limit the time involved in formally testing software solutions. Automated testing software is a category of software that can run a series of pre-determined tests on software solutions using test data and modelling keystrokes and mouse movements. Packages can be expensive, but make the testing process cheaper in the long term.

The costs of improper testing

Despite the fact that IT is a relatively young field in terms of world history, there have been some infamous examples of the effect that improper testing can have. The Therac-25 was a medical computer used to control radiation therapy. A bug in its code caused at least two deaths between 1985 and 1987 due to massive exposures to radiation (over 100 times the intended dose). A bug in the guidance system of the European rocket Ariane-5 caused the rocket to explode one minute after take-off, costing over one billion US dollars. The software within the guidance system attempted to store a 64-bit number in a memory location of only 16-bits.

Types of errors

Before discussing how we can test a module or coded solution for errors, it is useful to have an understanding of the different types of errors that can occur. Generally speaking, errors are classified as syntax errors, logic errors or run-time errors.

Syntax errors

Syntax errors are errors with the spelling or format of specific commands. As such, they do not occur very frequently in finished programs as most good programming language interfaces will detect them in the development stage. Compiled languages check the syntax of the entire code before they execute the program.

Logic errors

Logic errors are much harder to detect than syntax errors as they may not occur every time the program is run and they do not cause the program to crash. They occur when data causes the program to operate in a way that was not intended. For example, the program may give an incorrect answer to a calculation or retrieve the wrong record from a database.

Run-time errors

Run-time errors (or exceptions) occur when something happens that was not planned for, causing the program to crash. Common examples of this sort of error are when an index is used that is outside the bounds of an array or a number is placed into a variable that has not been allocated enough space to hold it. What follows is a brief description of some of the most commonly occurring errors. Most of these errors are run-time errors, but some are logic errors.

Arithmetic overflow and underflow

Arithmetic overflow happens when the result of a calculation or process causes a variable to become too large for the location in which it is stored. Underflow can occur when a number is too small to be represented in the variable type being used. This may be hard to visualise, but as small numbers are represented as exponents, small numbers have an exponent which is a large negative number.

Memory leak

As programs create variables and allocate memory to resources, they consume more of the processing power of the computer on which they are running. If these resources are not 'freed' up when they are no longer needed by the program, there is the potential for the program to increasingly use more and more memory. This may lead to reduced performance for all the programs being used at

the time and at its most extreme will cause the computer to crash. Many programming languages now include 'garbage collectors' whose sole function is to free up those areas of memory that are no longer needed by the program.

Handle leak

A similar type of error to the memory leak, a handle leak occurs when a program allocates a 'handle' to a resource (such as the screen, printer or other peripheral) and does not 'free' up this handle on completing the task. The program may then assign multiple handles to the same device, which will slow the computer down and may eventually cause a crash.

Buffer overflow

A buffer is a temporary memory location that is used to store data that has either been input or output before it is processed. Buffer overflows can occur when the data input into a program is not checked for size and is too big for the buffer to handle.

Stack overflow

A stack data structure is used by the CPU to keep track of the current line within the program. As subroutines are executed, the line number of the line calling the subroutine is placed in the data structure known as a stack, so that once it is finished, the CPU can move back to this point and execute the next line. If a program contains a circular reference to a subroutine (that is, a subroutine that in turn calls another that contains a call to the original subroutine), then the stack data structure can quickly fill up with line number references and overflow causing a program crash.

Deadlock

A deadlock can occur when two or more processes are waiting for each other to finish and so neither one does. Deadlocks can be very difficult to predict as they generally happen in multitasking or networked environments when different programs are sharing resources.

Off-by-one error

This type of error is a very common one in which a loop within a program either loops one too few or one too many times. This may or may not cause the program to crash, instead possibly causing an output or function to be incorrect. Often the cause for this problem is a programmer failing to take into account that an array begins at '0' as opposed to '1'. This error can also be caused by a comparison that is out by one (for example, 'less than' instead of 'less than or equal to').

Race hazard

A race hazard can occur when one process that is dependent upon the output from another process, executes before that data is received.

Type conversion

Type conversion errors can occur when data is changed from one data type to another. The conversion may be one which is not allowed, which will cause the program to crash. Alternatively, the data may lose accuracy and effectively become a different number.

Preventing or handling bugs

Bugs can be prevented or handled in a variety of ways. During design time, there are ways in which programs can be tested or checked for bugs. In addition to this, many programming language interfaces contain features that are designed to minimise the instance of errors. Some programming languages will also have syntax designed not to let programmers perform tasks that could potentially lead to problems. Lastly, all programs should contain some sort of error trapping routine, so that an unexpected error does not crash the program, but instead reports some information about why it has occurred.

Although the term 'bug' has its origins in engineering, the first computer 'bug' was found in a relay computer at Harvard University in 1945. Technicians opened up the computer to try to determine what was going

wrong, only to find a dead moth in-between one of the relays! They affixed the bug to the error log and wrote underneath "First actual case of bug being found". The original log (complete with the moth taped to it) can be found in the Naval Surface Warfare Center Computer Museum at Dahlgren, Virginia, USA. The technicians were reputedly the first to coin the term 'debugged'.

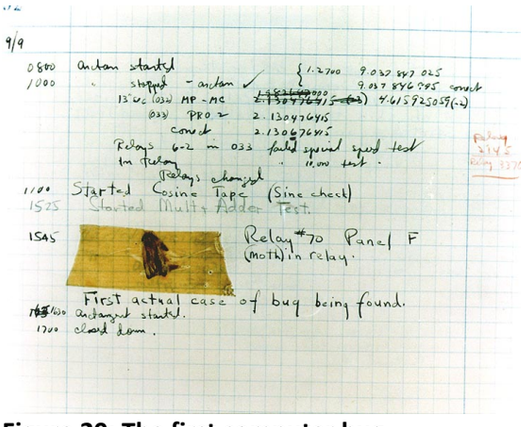


Figure 20: The first computer bug

As we are examining the stages of software development, we will focus on the methods which we can use to detect errors before the program is implemented.

Methods of testing

A trace table is a relatively easy tool to implement for a small algorithm or module (ideally before it is coded). While it is a tool that can also be used to test out the function of a coded solution, its focus is on specific logic and so it can be a little too fiddly to use on a larger scale.

To create a trace table, you must first look through the algorithm or coded solution and identify all of the main variables. These should be placed in a table like the one in Figure 21. Once this has been done, go through the algorithm line by line, and as the variables change, you should write their new values under the relevant column. You should also leave a space at the bottom of the table for anything that is output to the screen.

Note that columns are only modified when the value of a variable is changed. This way, when reading the trace table, the current state of the module or coded solution can be found by examining the bottom row. If a variable does not have a value entered in this row, the current stored value of the variable is the bottom most one.

A similar technique to creating a trace table is the use of 'watches' on key variables.

Watches and breakpoints

Most programming language interfaces allow you to set breakpoints in the code at which point the execution of the code will pause and allow you to examine it. Once the program has been paused in this way, it is usually possible to 'step' through the code a line at a time so that you can observe what is happening. 'Watches' can also be set up so that the value of certain variables is displayed on the screen while the program is executed.

Techniques like these are best when testing a module to ensure that it works within design

Trace table

ID	Surname	Tax_Rate	Done	Discount	Temp
001	Phillips	35	False	0	220
			True		300
002	Jones	42	False	32	15
				16	200
			True		220

Output (screen):
Customer: Phillips. Total Amount: $534.20
Customer: Jones. Total Amount: $320.50

Figure 21: Trace table example

specifications. Once focus moves to the testing of the coded solution as a whole, the use of trace tables and watch lists becomes hard as the number of variables can be extremely large.

Testing will be further discussed in Chapter 8, where we will focus our attention on the ways in which solutions are tested and evaluated.

Procedures and techniques for handling and managing files

A number of procedures and techniques are often used when managing files. As files are easily corrupted, it is often best to leave them 'open' for the shortest time possible. From a programming perspective the best way to do this is to 'open' the file, read in its contents and then 'close' the file. If the file is 'open' and the software solution experiences an unexpected crash, it is quite likely that the file will be damaged. Opening files for prolonged periods can also mean that other users cannot access the file.

Input files should be located in a folder that is specific to the software solution so that they are easy to locate and are grouped with the solution when files are backed up. This can also be achieved by including an import feature in the software that copies a file from any location to a secure folder within the solution.

Naming of files is also important. A specific file extension may be chosen so that files are not misinterpreted as belonging to another software package. Files should be named in ways that identify them and distinguish them from previous versions.

Common file formats

TXT file format

While all files are technically saved in a plain text format, a TXT file has no special structure to it (other than one that the software developer imposes via their code). It is very common for software developers to save small collections of data (such variable defaults or program statistics) into a TXT file. In doing so, the software developer will be ensuring that they write information and read information in the same order and insert any separators that they require. This lack of structure often means that software developers will instead opt for one of the two other formats listed next.

CSV file format

One of the most common file formats is a CSV or Comma Separated Value format. As the name suggests, a CSV file is a text file in which the items of data are separated by commas. Many applications are able to produce CSV files as their output and accept CSV files for input. A CSV file can fairly easily replicate what is held in a spreadsheet as its format and structure is very similar. Though it is more common to encounter a comma delimited file, the character that is placed between the different data items can be varied. If a software solution is using the file exclusively, then the choice of character (known as the delimiter) does not matter. However, if the data is going to be passed between software packages, the choice of delimiter is obviously very important.

XML file format

XML or Extensible Markup Language is a method of encoding data in a file that is both well organised and accessible for software solutions as well as easy to read. It is standards based and used extensively by software developers, especially when sharing of data over the Internet is required. It shares a similar structure to HTML, which also uses 'markups' in the form of tags in its structure. One of the most useful aspects of representing data in XML file format, is that the format itself defines the data and how it is structured. This is in contrast to a CSV file where the exact number and data types would need to be known in advance for a software package to read them.

An example of an XML file is shown on the next page.

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog>
        <movie>
        <title>Star Wars</title>
        <director>George Lucas</director>
        <writer>George Lucas</writer>
        <year>1977</year>
        </movie>
        <movie>
        <title>The Shawshank
        Redemption</title>
        <director>Frank
        Darabont</director>
        <writer>Stephen King</writer>
        <year>1994</year>
        </movie>
</catalog>
</xml>
```

Figure 22: XML file example

As you can see in Figure 22, the structure of an XML file is similar to that of a flat file database. The fields are also well defined, unlike in a CSV file. Also of importance when discussing the handling and management of files are issues of security, archiving, backing up and disposal.

Searching for data in an array

A common programming task is searching for data in an array. The simplest way to do this is to examine each item in the list in turn, until the required item is found. This is known as a linear search. Although it is not very efficient, it is very easy to code and works well for small lists. For a list of 1000 items, a linear search could take up to 1000 comparisons to find the required item. On average, it will take 500 comparisons. Figure 23 shows an example of a linear search.

In the linear search algorithm shown, 'subFind' is set to '-1' before the loop starts. As the value of 'subFind' will be changed to be the index of the item that is being searched for, if it is not found, 'subFind' will still be equal to '-1' at the end of the search. This can be used to indicate that the item is not in the list. Note also that the 'Found' flag is used to end the loop if the item is found, rather than allow the loop to check all of the remaining array elements.

A better search method is called a binary search.

Linear search algorithm

Variables
Integer array: List[ ] – will hold the items in the array.
Integer: Count – will hold the index of the array element we are examining.
Integer: Max – will hold the size of the array.
Integer: Item – will hold the item that we are searching for.
Boolean: Found – indicates if the item has been found or not.

```
Subroutine subFind(Item, List, Max)
Begin
    subFind ← -1
    Count ← 0
    Found ← False
    While (Count <= Max and not Found)
        If List[Count] = Item Then
            subFind ← Count
            Found ← True
        Else
            Count ← Count + 1
        End If
    End While
End
```

Figure 23: Linear search algorithm

Binary search

A binary search is a search that can be carried out on a sorted list. It works by dividing a list in half each time a comparison is made. As a result, it is extremely fast, although if the list is not in sorted order already, this does mean an additional overhead. The best policy is if you plan to utilise a binary search in your program, keep your data in sorted order at all times, which then reduces this overhead considerably.

Let's examine how a binary search would locate some items contained in a list.

Binary search example

Let's examine how a binary search would locate some items contained the list below:

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
List(Index)	3	5	8	9	12	17	27	28	30	42	44	47	50

Note that the array shown above has already been sorted.

We will start by searching for the number '12'. Two variables - 'Low' and 'High' will be used to keep track of the two ends of the list we are examining. Another variable – 'Middle' will be used to point to the item we are comparing to our search item.

At the beginning of the search, 'Low' is set to the index of the start of the list (in this case '0') and 'High' is set to the index of the end of the list (in this case '12'). 'Middle' is set to the middle index between 'Low' and 'High' (rounded down). If 'Middle' is the item we are looking for, the search is finished. In this case, it is not, so we continue to the next comparison. The variables for comparison '1' are shown below:

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
List(Index)	3	5	8	9	12	17	27	28	30	42	44	47	50
	L						M						H

If the item we are looking for is smaller than the item at position 'Middle' then we set 'High' to be equal to 'Middle' minus one. If it is larger, then 'Low' is set to be equal to 'Middle' plus one. This way, we have halved the list. 'Middle' is reset to the mid-point between 'Low' and 'High'. The variables for the second comparison are shown below:

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
List(Index)	3	5	8	9	12	17	27	28	30	42	44	47	50
	L		M			H							

Note that 'Middle' has been set to '2' as it is the value of 'High' – 'Low' divided by 2 and rounded down.

Once again, the item at 'Middle' is not the item we are looking for. As our search item is higher, we set 'Low' to be equal to 'Middle' plus one. We also reset 'Middle' to the mid-point between 'Low' and 'High'. The variables for the third comparison are shown below:

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
List(Index)	3	5	8	9	12	17	27	28	30	42	44	47	50
				L	M	H							

'Middle' is now pointing to the item we are searching for, so we are now finished. It has taken three comparisons in this case. Using a linear search method, on average it would take 6.5 comparisons. A binary search would take a maximum of 10 comparisons to find an item in a list of 1,000 items, and a maximum of 20 comparisons to find an item in a list of 1,000,000! For those that are mathematically minded, the formula to work out the number of comparisons is log n (base 2) + 1 (where n is the number of items in the array).

Figure 24: Binary search example

Binary search algorithm

Variables

Integer array: List[ ] – will hold the items in the array.
Integer: Low – will hold the index of the start of the list we are examining.
Integer: Max – will hold the size of the array.
Integer: High – will hold the index of the end of the list we are examining (Max – 1)
Integer: Middle – will hold the index of the middle of the list we are examining (half way between 'Low' and 'High').
Integer: Item – will hold the item that we are searching for.
Boolean: Found – indicates whether the item has been found or not.

```
Subroutine subFind(Item, Max)
Begin
        Low ← 0
        High ← Max - 1
        subFind ← -1
        Found ← False
        While (Low <= High and not Found)
                Middle ← (Low + High) Div 2
                If List[Middle] = Item Then
                        subFind ← Middle
                        Found ← True
                End If
                If List(Middle) > Item Then
                        High ← Middle – 1
                Else
                        Low ← Middle + 1
                End If
        End While
End
```

The 'Div' function gives the whole number result of one number divided by another

Figure 25: Binary search algorithm

In the binary search algorithm shown above, 'subFind' is set to '-1' before the loop starts. As the value of 'subFind' will be changed to be the index of the item that is being searched for, if it is not found, 'subFind' will still be equal to '-1' at the end of the search. This can be used to indicate that the item is not in the list, although the Boolean variable 'Found' is also used for this purpose. The way that the algorithm 'knows' that it has reached the end of the search, is when 'Low' becomes greater than 'High' – which will happen if 'Low' and 'High' are the same value and the item that is being examined is not the item that is being searched for.

Sorting data in an array

It is a common programming task to incorporate a sorting routine into a program. Many programming environments have incorporated sorting routines and these can be sufficient for the types of solutions we are building. There can be disadvantages to using these, however. When making use of inbuilt sorting routines, we have no idea whether they are very efficient or how they have been coded. An understanding of how sorting routines work gives the software developer the ability to code their own, especially in cases

when the data set being sorted is particularly large and efficiency gains are required.

There are a large number of different sorting routines. A quick search of the Internet will uncover many of these – and will probably also raise the question: Why so many? Each sorting routine that has been developed has resulted in gains in the number of comparisons within the routine and/or the number of times array elements need to be swapped around – ultimately leading to a quicker result. Most sorting routines are designed to move the array elements around rather than copy the array to another place in memory. This can be important when the size of the array to be sorted is very large. Sorting algorithms will almost always utilise a small number of temporary variables so that swaps can be made.

For the purposes of this study, we will examine two sorting routines. The array of data shown in Figure 26 below will be used to compare each of them. Note that the data set starts at array index '0'.

examining each item in the array and finding the smallest one (in the case of sorting from lowest to highest). Once found, this item is swapped with the item at the start of the array. The sort then examines the items from the next position to the end of the array, looking again for the smallest item. Once found, it is swapped with the item at the second position. Each subsequent pass gets smaller by one – as the sorted portion of the array gets larger.

A selection sort is an easy sort to write and understand. Despite its relative simplicity, it is not a bad sorting routine and can perform quite adequately if the situation is right.

Selection sorts do a large number of comparisons (of the order of n^2 where n is the number of array elements being sorted). One of the things that is very consistent about a selection sort is the number of swaps and passes it will do. It will always perform n-1 swaps as well as n-1 passes.

Some sorts have what is known as an 'exit clause'. Some sorting algorithms are able to

Array of data for sort comparison

Index	0	1	2	3	4	5	6	7	8	9
List(Index)	35	20	15	30	28	45	17	3	49	32

Figure 26: Array of data for sort comparison

Each time the array is cycled through, we call this a 'pass'. Each time two array elements are compared to each other, we call this a 'comparison'. Each time two array elements are swapped, we call this a 'swap'. Each of these is an important measure in determining if a sorting routine is performing well.

Let's look at our first sorting routine (also one of the simplest ones) known as a selection sort.

Selection sort

The selection sort was one of the first sorting routines and it attempts to sort the data in a very 'human' way. A selection sort starts by

detect whether an array has become sorted and then exit the sort earlier. A selection sort is not able to do this. In fact, an array that is already sorted will take just as long to 'sort' using a selection sort as one that is in reverse order (for example). Many of the more sophisticated sorting routines move the array elements towards a sorted state in the process of doing simple comparisons – which a selection sort does not do.

Let's look at our array of data and use a selection sort to arrange it in ascending order.

Selection sort example part 1

Below is the array of data that we will be using as a starting point for our sorting comparisons.

Index	0	1	2	3	4	5	6	7	8	9
List(Index)	35	20	15	30	28	45	17	3	49	32

On our first pass through the array, we look for the smallest number – which is '3'.

Index	0	1	2	3	4	5	6	7	8	9
List(Index)	20	35	15	30	28	45	17	**3**	49	32

We have now identified which number needs to go at the beginning of the array. So all we need to do is swap it with the number that is there ('20') and our first pass is complete.

End of Pass 1

Index	0	1	2	3	4	5	6	7	8	9
List(Index)	**3**	35	15	30	28	45	17	**20**	49	32

Now we can start the second pass through the array. As we now have the first element in place, we can start our loop from position '1' instead of '0'. Again we are looking for the smallest element – which this time is '15'.

Index	0	1	2	3	4	5	6	7	8	9
List(Index)	3	35	**15**	30	28	45	17	20	49	32

To end the second pass, we swap this element into its correct place in element '1'.

End of Pass 2

Index	0	1	2	3	4	5	6	7	8	9
List(Index)	3	**15**	**35**	30	28	45	17	20	49	32

Let's do the rest of the passes in quick succession, now that you can see how each pass is done.

End of Pass 3

Index	0	1	2	3	4	5	6	7	8	9
List(Index)	3	15	**17**	30	28	45	**35**	20	49	32

End of Pass 4

Index	0	1	2	3	4	5	6	7	8	9
List(Index)	3	15	17	**20**	28	45	35	**30**	49	32

End of Pass 5

Index	0	1	2	3	4	5	6	7	8	9
List(Index)	3	15	17	20	**28**	45	35	30	49	32

Figure 27a: Selection sort example part 1

Selection sort example part 2

End of Pass 6

Index	0	1	2	3	4	5	6	7	8	9
List(Index)	3	15	17	20	28	**30**	35	**45**	49	32

End of Pass 7

Index	0	1	2	3	4	5	6	7	8	9
List(Index)	3	15	17	20	28	30	**32**	45	49	**35**

End of Pass 8

Index	0	1	2	3	4	5	6	7	8	9
List(Index)	3	15	17	20	28	30	32	**35**	49	**45**

In the final pass of a selection sort, the last two array elements are compared and swapped (if required). This then completes the sort.

End of Pass 9

Index	0	1	2	3	4	5	6	7	8	9
List(Index)	3	15	17	20	28	30	32	35	**45**	**49**

For an array of 10 elements, a selection sort will execute 9 passes. In general terms, for an array of size 'n', a selection sort will execute 'n-1' passes.

Figure 27b: Selection sort example part 2

Let's discuss a typical algorithm for a selection sort

The algorithm shown in Figure 29 on the next page works by using two nested loops. The first loop 'PassNumber' will perform the passes (1 less than the total number of array elements) and the second loop 'Position' will cycle through the array elements. As the second loop is executed, it compares the current array element to the smallest value for this pass. If it is the smallest value found, it is recorded as well as its array position. The second loop is designed to get progressively smaller, as each pass through the list will mean that an element is placed in its correct position at the beginning of the array.

The algorithm also utilises a simple routine to swap two elements, as shown in Figure 28.

The 'swap' subroutine

Variables

Integer: Temp – will hold one of the list items while it is being swapped.
Integer: Index1 – the index of the first list item to be swapped.
Integer: Index2 – the index of the second item.

Subroutine subSwap(Index1, Index2)
Begin
    Temp ← List(Index1)
    List(Index1) ← List(Index2)
    List(Index2) ← Temp
End

Figure 28: 'swap' subroutine

Selection sort algorithm

Variables

Integer array: List[ ] – will hold the items in the array.
Integer: ListSize – the number of items in the array.
Integer: PassNumber – the counter for the first loop.
Integer: Position – the counter for the second loop.
Integer: Smallest – will hold the value of the smallest array element found in the current pass.
Integer: SmallestPos – will hold the index of the smallest array element found.

```
Subroutine subSelectionSort(List, ListSize)
Begin
        Loop PassNumber from 1 to ListSize – 2
                Smallest ← List(PassNumber – 1)
                SmallestPos ← PassNumber
                Loop Position from PassNumber - 1 to ListSize - 1
                        If List(Position + 1) < Smallest Then
                                Smallest ← List(Position + 1)
                                SmallestPos ← Position + 1
                        End If
                Next Position Loop
                Swap(PassNumber - 1, SmallestPos)
        Next PassNumber Loop
End
```

Figure 29: Selection sort algorithm

Quick sort

A quick sort is a popular sorting algorithm as it is both efficient and fast. The most popular versions of the quick sort algorithm make use of a programming method known as recursion.

Let's firstly look at how a quick sort works. Note that there are also versions of this sort that use the 'pivot' value in different ways. The version detailed in Figure 31 uses a pivot value at the start of the array (or sub-list).

A quick sort starts by setting a 'pivot' value which is the item at the start of the list. As this routine works by examining smaller and smaller lists, we will define variables to keep track of the current 'start' and 'end' of the list we are examining.

On the first pass through the array, items in the list are compared to the 'pivot' value. We will keep track of what items we are comparing by using two variables 'up' and 'down'. 'Up' will begin at the 'start' of the list and work up and 'down' will start at the 'end' of the list and work down. If the item that 'up' is pointing to is larger than the 'pivot', the idea is to swap it with an item that is smaller than the 'pivot' being pointed to by 'down'. The first pass is complete when 'down' becomes smaller than 'up'. At this point, the 'pivot' value is swapped with 'down' and should then be in the correct place in the array.

Let's see how this works in practice.

Quick sort example part 1

Here is our array of data that we used in the selection sort example.

Index	0	1	2	3	4	5	6	7	8	9
List(Index)	35	20	15	30	28	45	17	3	49	32

For the array above, 'start' and 'up' are set to '0' and 'end' and 'down' are set to '9'. This would make the 'pivot' value equal to '35', which is the value at 'start'. As we move through the array, we will be comparing items to '35'.

Let's begin. The value at 'up' is compared to the 'pivot' value. In this case '35' is equal to '35' and so we will leave it in place. 'Up' is incremented and the next value is compared to '35'. '20' is less than '35' so leave it in place. 'Up' is incremented again. '15' is also less than '35'. We keep incrementing 'up' until we find a value that is greater than '35'. This happens when 'up' is pointing to index '5' which is '45'. We have now found our first value to swap. 'Down' is now moved down the list. '32' is less than '35' which means we have our second value to swap. '32' and '45' are now swapped.

Index	0	1	2	3	4	5	6	7	8	9
List(Index)	35	20	15	30	28	**32**	17	3	49	**45**

'Up' now continues to move up the list till it finds a value that is greater than the pivot. '49' is greater than the pivot, so 'up' stops at index '8'. 'Down' is moved down the list until it finds a value that is less than the pivot, or it becomes smaller than 'up'. 'Down' does become smaller than 'up' when it has a value of '7'. When this happens, this signals that the pass is over. The last thing to do is to swap the 'pivot' value with 'down'. '35' is swapped with '3' and the array looks like the one below.

End of Pass 1

Index	0	1	2	3	4	5	6	7	8	9
List(Index)	**3**	20	15	30	28	32	17	**35**	49	45

The 'pivot' is now in its correct place in the array. You will notice that all the values below the 'pivot' are smaller and all the values above the 'pivot' are larger.

We now divide the list into two sub-lists, each of which we sort using the same routine. The sub-lists are defined as being either side of the 'pivot' value (pointed to by 'down'). If a sub-list consists of one or no elements, we consider it sorted. In this case, we would have two sub-lists. The first list would start at '0' and go to 'down' – 1 (which in this case is '6'). The second sub-list would start at 'down' + 1 (which is '8') and go to the 'end' of the list.

As the quick sort routine works by sorting smaller and smaller sub-lists, we will consider the completion of each sub-list to be a pass. Let's sort the second sub-list first. The 'start' of this list would be '8' and the 'end' would be '9'. As this list has only two elements, we only need to sort these. In this case, swapping '49' and '45' completes the sorting of this sub-list.

End of Pass 2

Index	0	1	2	3	4	5	6	7	8	9
List(Index)	3	20	15	30	28	32	17	35	**45**	**49**

Figure 30a: Quick sort example part 1

Quick sort example part 2

Let's now look at the first sub-list. The 'start' of this sub-list is '0' (which makes the 'pivot' equal to '3') and the 'end' is '6'. 'Up' is set to '0' and 'down' is set to '6'. 'Up' is now moved up the list until it finds a value that is greater than the 'pivot' – which it does at index '1'. 'Down' is now moved down the list till it finds a value that is less than the 'pivot' – which it does not do. 'Down' now has a value of '0' and as it is less than 'up', the 'pivot' is swapped with 'down' (which is not really a swap at all).

End of Pass 3

Index	0	1	2	3	4	5	6	7	8	9
List(Index)	3	20	15	30	28	32	17	35	45	49

As before, we try to split what is either side of the 'pivot' into two sub-lists. The first sub-list will be from 'start' to 'down' – 1. This is a list of zero length, so we can ignore it. The second sub-list will be from 'down' + 1 (which is '1') to 'end' (which is '6').

Once again, 'start' is set to the start of this list ('1') and 'end' is set to the end of this list ('6'). The 'pivot' value in this case is '20'. 'Up' is set to '1' and 'down' is set to '6'. 'Up' begins to move up the list again till it finds a value that is greater than the 'pivot'. This happens when 'up' is '3' (which contains the value '30'). 'Down' is now moved down the list till it finds a value that is less than the 'pivot', which happens immediately. '30' and '17' are swapped. 'Up' continues to move up the list, this time stopping at '28'. 'Down' continues to move down and passes 'up'. At this stage, 'up' has a value of '4' and 'down' has a value of '3'. As 'down' is now less than 'up', the 'pivot' and 'down' are swapped which places the 'pivot' in the correct place and ends the current pass.

End of Pass 4

Index	0	1	2	3	4	5	6	7	8	9
List(Index)	3	17	15	20	28	32	30	35	45	49

Hopefully now this process is clearer to you. To speed the process up, the remaining passes will be shown to you – but I would encourage you to work them out for yourself first to check your understanding.

End of Pass 5

Index	0	1	2	3	4	5	6	7	8	9
List(Index)	3	17	15	20	28	32	30	35	45	49

End of Pass 6

Index	0	1	2	3	4	5	6	7	8	9
List(Index)	3	17	15	20	28	30	32	35	45	49

End of Pass 7

Index	0	1	2	3	4	5	6	7	8	9
List(Index)	3	15	17	20	28	30	32	35	45	49

The list is now sorted. With a small list such as this, the size of the sub-lists was sometimes very small (1 or 2 array elements). With larger lists, the sub-lists are divided and further divided and the sort works backwards once it reaches these finishing points.

Figure 30b: Quick sort example part 2

Figure 31 below shows an algorithm for a quick sort (using recursion). Before we discuss how it works, let's touch on the topic of recursion.

Recursion is the process by which a function or procedure calls itself. This can be a very useful tool as it can make a complex process simpler and easier to code. A recursive procedure should always have an end point, after which time it will begin to work its way back through the many procedure calls that have been made. Now that you know how a quick sort works, you can see that it divides the array into sub-lists, each of which is sorted using the quick sort routine.

Quick sort algorithm

Variables

Integer array: List[ ] – will hold the items in the array.
Integer: Start – will hold the index of the left most element in the sub-list that is being examined.
Integer: End – will hold the index of the right most element in the sub-list that is being examined.
Integer: Pivot – the pivot value for the list or sub-list.

```
Function Quicksort(List)
Begin
        If length(array) > 1 Then
                Pivot ← first element of the array List
                Start ← first index of List
                End ← last index of List
                While Start <= End
                        While List(Start) < Pivot
                                Start ← Start + 1
                        End While
                        While List(End) > Pivot
                                End ← End -1
                        End While
                        If Start <= End Then
                                Swap List(Start) with List(End)
                                Start ← Start + 1
                                End ← End – 1
                        End If
                End While
                Quicksort(List from first index to End)
                Quicksort(List from Start to last index)
        End If
End
```

Figure 31: Quick sort algorithm

This algorithm is quite complex and hard to understand by simply reading through it. To come to grips with the way that it works, perhaps put some test data together and perform a desk check of the algorithm. Alternatively, code the algorithm and test it out with some large unsorted lists to see what happens and how quickly the list is able to be sorted.

Context Questions

1. What condition must be met for a binary search algorithm to be used successfully on a collection of data?
2. Define the terms ‘pass’, ‘comparison’ and ‘swap’.
3. Of the terms above, which do you think would have the greatest negative effect on the efficiency of a sorting algorithm?
4. If the program shown in Figure 15 were redesigned to read one million numbers into a suitably sized array, how many additional lines of code would be required?
5. What is a data structure?
6. Why does the index of a 1D array typically begin at zero?
7. An array has been declared as having a maximum index of 10. How many elements does it consist of?
8. What advantages does the use of arrays have when designing programs that deal with large quantities of data?
9. What is a record?
10. Explain the term ‘Garbage In, Garbage Out’.
11. Describe ways in which a well designed user interface can minimise the amount of validation code that is required.
12. Explain why it is logical to perform an existence check before a type check.
13. Suggest a suitable range check that could be used on a person’s birth date.

Applying the Concepts

- Create a program of your own that will generate a list of 10 thousand random numbers and then sort them using a selection sort and a quick sort. It should display the time and number of passes once sorted.

Key Skills Checklist

At the conclusion of this chapter, you should be able to address the following key skills. Mark each off as you can achieve them.

Understand how one-dimensional arrays can be used to store data	☐
Explain the differences between associative arrays and records	☐
Compare the differences between TXT, CSV and XML file formats	☐
Explain the difference between subroutines, functions, events and methods	☐
Describe the process of undertaking a selection or quick sort	☐
Explain how a linear search is conducted and what must be done to use a binary search	☐
Describe how data can be validated using existence checking, range checking and type checking	☐
Explain how test data can be constructed to test that modules work correctly	☐

Sample Examination Questions

The following sample examination questions can be attempted to test your knowledge of the content of this chapter.

Question 1
A hash function can be used to:

- **A.** sort data
- **B.** make data in a list easier to find
- **C.** remove data that is no longer current
- **D.** create a copy of a set of data

Question 2
The validation process known as type checking is:

- **A.** Used to prevent data of the incorrect variable type being accepted as input
- **B.** Only effective if used as a method after a range check
- **C.** Is used to check to see if a number has been entered with the correct number of decimals
- **D.** Is only applicable when considering postcodes

Question 3
A difference between CSV and XML file formats is:

- **A.** XML files are much harder to visually read than CSV files
- **B.** CSV files do not contain any delimiters
- **C.** CSV files are much larger than XML files for the same data set
- **D.** XML contains tags which define field names and structure

The following information is required for Questions 4, 5, 6 and 7.

Consider the following array of data:

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
List(Index)	33	29	76	32	3	17	91	28	30	85	19	64	52

Question 4
If the array above were to be searched using a linear search method, the maximum number of comparisons to find an item would be:

- **A.** 5
- **B.** 12
- **C.** 13
- **D.** 11

Question 5
For the array above to be searched using a binary search, it must first be:

- **A.** Copied to another array
- **B.** Sorted
- **C.** Placed into a hash table
- **D.** Analysed for duplicates

Question 6
If the array of data was prepared correctly for a binary search, the maximum number of comparisons to find an item would be:

A. 1
B. 3
C. 4
D. 5

Question 7
If the array above was prepared correctly for a binary search, the maximum number of comparisons to determine if an item was not present would be:

A. 1
B. 3
C. 4
D. 5

Question 8
After being alerted to an issue by a customer, Emily examines some code in a software solution that has been published by one of her employees. Emily is very surprised to find a number of syntax errors as well a large logic error in a financial year tax calculation.

The logic error would most likely have meant that when the software was run without syntax errors:

A. It would have crashed
B. It would not have performed the calculation at all
C. It would have run a number of times and then stopped
D. It would have given the incorrect results to the tax calculation

The following information is required for Questions 9, 10 and 11.

```
▼<CATALOG>
  ▼<CD>
     <TITLE>Empire Burlesque</TITLE>
     <ARTIST>Bob Dylan</ARTIST>
     <COUNTRY>USA</COUNTRY>
     <COMPANY>Columbia</COMPANY>
     <PRICE>10.90</PRICE>
     <YEAR>1985</YEAR>
   </CD>
  ▼<CD>
     <TITLE>Hide your heart</TITLE>
     <ARTIST>Bonnie Tyler</ARTIST>
     <COUNTRY>UK</COUNTRY>
     <COMPANY>CBS Records</COMPANY>
     <PRICE>9.90</PRICE>
     <YEAR>1988</YEAR>
   </CD>
  ▼<CD>
     <TITLE>Greatest Hits</TITLE>
     <ARTIST>Dolly Parton</ARTIST>
     <COUNTRY>USA</COUNTRY>
     <COMPANY>RCA</COMPANY>
     <PRICE>9.90</PRICE>
     <YEAR>1982</YEAR>
   </CD>
```

Question 9
The data shown above is in which file format?

A. CSV
B. HTML
C. Javascript
D. XML

Question 10
This is not a complete file. It is easy tell this because:

- **A.** There is no closing 'catalog' tag
- **B.** The last item listed begins with 'G' and this is not close to the end of the alphabet
- **C.** The file states that there are 7 items in total
- **D.** The arrow at the side of the main tag has not been expanded

Question 11
The <YEAR> field would most likely be of type:

- **A.** Character
- **B.** Text
- **C.** Number
- **D.** Date

Question 12
Client.Surname could be an example of a piece of code you might see when dealing with:

- **A.** a one dimensional array
- **B.** a hash table
- **C.** a data dictionary
- **D.** a record

Question 13
When coding a search for data in an array, Joey decides to start at the beginning and check each array element in turn. This is called a:

- **A.** linear search
- **B.** brute force
- **C.** binary search
- **D.** hash function

Question 14
What is a trace table and what is its' main purpose?

_____

_____

2 marks

Question 15
Consider the following array of numbers that is to be sorted in ascending order using a selection sort. The sort is programmed to move through the array from left to right.

Index	0	1	2	3	4
List(Index)	65	23	45	62	27

a. On the first pass through the array, what number would be selected?

Value: _____

1 mark

b. How many passes will it take to sort this array?

Value: _____

1 mark

c. If this array had 1000 elements, how many passes would it take to sort this array?

Value: _____

1 mark

d. If this array had 1000 elements, how many comparisons would it take to sort this array?

Value: _____

1 mark

Sample Examination Answers

Question 1
Answer: B

A hash function on its own does not sort data. Best response in this case is option B.

Question 2
Answer: A

Question 3
Answer: D

Question 4
Answer: C

Must consider the possibility that the item is not contained in the array at all.

Question 5
Answer: B

Question 6
Answer: C

Question 7
Answer: C

Answer to Q6 and Q7 is the same. It takes the same number of comparisons to determine if a number is present or not.

Question 8
Answer: D

Question 9
Answer: D

Question 10
Answer: A

Question 11
Answer: C

Question 12
Answer: D

Question 13
Answer: A

Question 14
A trace table is a way of recording the current values of variables in an algorithm as it is tested. The main purpose of a trace table is in testing – the benefit being that a software developer can see all of the values and how they change line by line.

Question 15

- **a.** 23
- **b.** 4
- **c.** 999
- **d.** 499,500

Chapter 4
From planning to analysis

The chapter covers Unit 3: Area of Study 2 key knowledge:

- *Data and information*
    - *KK2.2 Techniques for collecting data to determine the needs and requirements, including interviews, observation, reports and surveys*
    - *KK2.3 Functional and non-functional requirements*
    - *KK2.4 Constraints that influence solutions, including economic, legal, social, technical and usability*
    - *KK2.5 Factors that determine the scope of solutions*
    - *KK2.14 Development model approaches, including agile, spiral and waterfall*
    - *KK2.15 Features of project management using Gantt charts, including the identification and sequencing of tasks, time allocation, dependencies, milestones and critical path*
- *Interactions and impact*
    - *KK2.16 Goals and objectives of organisations and information systems*
    - *KK2.17 Key legal requirements relating to the ownership and privacy of data and information.*

Key terms: design brief, mission statement, goals, objectives, information system, quantitative, qualitative, solution requirements, functional requirements, non-functional requirements, constraints, scope, project management, tasks, resources, critical path, developmental models, waterfall, agile, spiral.

Study roadmap

	Unit 3		Unit 4	
Ch	Area of study 1 – Programming	Area of study 2 – Analysis and Design	Area of study 1 - Development and Evaluation	Area of study 2 - Cybersecurity: software security
1	✓			
2	✓			
3	✓			
4		✓		
5				
6				
7				
8				
9				
10				

Planning a new software solution

One of the first steps in moving forward with the creation of a new software solution is to plan the various stages and timelines so that the scope of the project can be better understood. The process of doing this is often placed under the heading of 'project management'. Note that it is only really possible to undertake the planning of a project, if there is already an understanding of the aims of the proposed software solution. If this has not yet been determined, it may first be necessary to determine what the problems are within the information system and what may need to be done to solve them.

A design brief

A design brief is a simple document (or statement) that outlines what the problems are and anything about the nature of these problems that is currently understood. Note that a design brief serves to start the analysis process and is not an end product of analysis. It may not contain much detail at all.

One of the first things that a software developer needs to come to an understanding about is the goals and objectives of the organisation. It is also important to gain insight into the existing systems that are in place and how the proposed development outlined in the design brief, fits into the overall picture.

Organisations and goals

An organisation is an entity that has a collective goal (often stated broadly in a 'mission statement'). While organisations differ in size (from sole traders or individuals up to large multi-nationals), they can also differ in many other aspects such as:

- government / private (commercial)
- not for profit
- charities
- clubs and groups
- political parties.

While there is a lot that can be said about organisations and their structure, we are not going to explore this in depth. Instead our focus is on the goals and objectives of an organisation. As a software developer (and indeed as an employee of an organisation), it is vital that you have an understanding of the goals that the organisation has, how these translate into specific objectives and what information systems exist within the organisation to help them achieve these goals and objectives.

Mission statements

A mission statement is essentially a statement of the ultimate goals of the organisation. It is a statement of why the organisation exists. The mission statement guides the actions of the organisation and provides the framework through which all other strategy or decisions are formulated. Often the directors of organisations or the board of management will find themselves being drawn back to the mission statement when discussing new business decisions.

While there is a lot more to organisational goals and objectives than simply a mission statement, it is the starting point. A mission statement is often a public statement that can be found on an organisation's website.

For example, the mission statement for Microsoft is to create a family of devices and services for individuals and businesses that empower people around the globe at home, at work and on the go, for the activities they value most. While this is a very broad statement, it is the starting point for many other goals and objectives and is the core phrase that the organisation refers back to when making decisions.

Goals and objectives

Goals and objectives provide a framework to organisations to assist them to make strategic decisions. Whenever a new proposal or course of action is before the board of management for consideration, they will often refer back to the core goals and objectives for guidance. It is when these goals and objectives are ignored or forgotten about that organisations risk making decisions that not only put the business at risk, but can potentially change the way people perceive the company as a whole.

Goals are statements that describe a (potentially) future state or principle that the organisation strives to uphold or achieve. Objectives support the goals by providing some targets with measurable results.

Goals serve four basic purposes. They provide guidance and direction, assist planning for the future, inspire those within the organisation and support the evaluation process.

Typical examples of goals for organisations include the following things:

- profit (for commercial organisations only)
- achieving a specified percentage of market share
- service to a cause (for non-profits organisations)
- quality products
- excellent customer service (for non-profits this may well be their main goal)
- excellent reputation
- the price and reliability of products
- the care and development of staff.

As goals are very broad statements (generally), it can be difficult for an organisation to know the best way to act on them. In addition to this, it is important to be able to reflect on and assess the success of these goals – that's where organisational objectives come into play.

Organisational objectives are often quantifiable statements that expand upon the goals of an organisation in a way that allows them to be assessed.

For example, some examples of organisational objectives might be:

- increase sales in the next financial year by 10%
- reduce the number of customer complaints by 50%
- ensure that 95% of products ordered online are packaged and delivered within 5 working days
- maintain an employee retention rate of 95%
- engage 3 new clients per quarter

Information systems

The term 'information system' encompasses the combination of people, procedures, equipment and data that process data and information. It is an important definition as it is easy to think of an 'information system' as simply consisting of hardware and software. The processes that are used by the organisation, the data within the system and the roles that people play are just as important (if not more so) and have a significant impact on the creation of a software solution.

> *Information system: the combination of digital hardware, software and network components (digital systems), data, processes and people that interact to create, control and communicate ideas and data in digital solutions.*
>
> *VCAA VCE Computing Study Design 2020-2023 Glossary*

Information systems within organisations support the goals and objectives of the organisation. If an information system is not doing this or not doing it well, then its function needs to be re-evaluated.

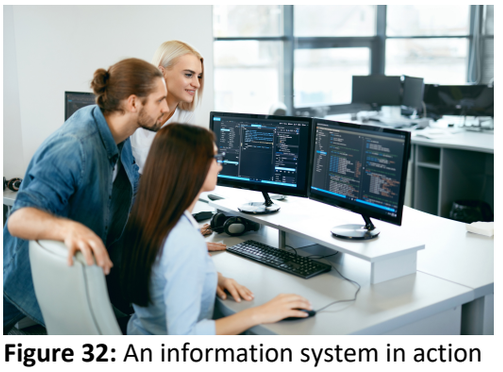


Figure 32: An information system in action

Goals and objectives of information systems

In order to support the goals and objectives within an organisation, information systems have their own goals and objectives. These goals and objectives are expressed in a very similar style to those of the organisation as a whole.

Some examples of information system goals are shown on the next page.

- Keep track of all of the employee payroll information and issue payments on time.
- Record customer transactions, manage stock and issue receipts.
- Record attendance and manage class rolls.
- Produce quarterly reports of sales.

You will notice that these goals are similar in style to the overall goals an organisation will have. The key difference between organisational and information system goals, is organisational goals describe the overall goals that the organisation has and information system goals describe what each system within the organisation is trying to achieve.

Information system objectives perform the same purpose as organisational objectives. Some examples of information system objectives are shown below:

- Ensure that all data is securely archived and that no data is lost.
- Provide a response time to user queries that is acceptable to all users.
- Perform in a stable and reliable fashion for 99.5% of the time (up-time).
- Process up to 500 reports per month.
- Store up to 10,000 transactions per week.
- Reduce the number of user generated validation errors by 50%.

Types of interactions (inputs and outputs) generated by information systems

The inputs and outputs from an information system will be determined by the type of information system concerned. Information systems are categorised by their purpose and function. The study of the different types of information system can be a detailed topic on its own and for that reason, we will simply examine how they differ functionally.

The information systems that deal primarily with the processing of transactions are grouped together as their function is similar. These systems are categorised by their ability to process transactions as they occur continuously. An example of such a system might be a ticketing system for a concert hall. The types of inputs and outputs processed by this system are:

- list of performances being offered (output)
- list of available dates for each performance (output)
- performance name (input)
- performance date (input)
- availability of tickets (output)
- categories of ticket available (output)
- price of tickets for each category (output)
- requested tickets (input)
- customer details (input)
- billing details (input)
- ticket confirmation (output).

The transactions that occur in such a system would be roughly in this order. You will also notice that the billing of the customer would possibly be handled by another system that just handles these processes (verifying credit cards, etc.) passing information to and from the ticketing system.

Another type of information system is used by the management of organisations to help them make decisions. An example of these sort of systems might those that assist with decisions such as whether a stock item is worth keeping in a store or how classes of students are performing in a subject from year to year.

Let's look at an example of an information system that is comparing the results of students in a range of subjects. A system such as this (or more correctly, many such systems), is used by VCAA to analyse student data and respond to the needs of students.

The types of inputs and outputs processed by this system are:

- student enrolments for the year (input)
- GAT results (input)
- SAC results (input)

- moderated scores (output)
- results of previous years (input)
- analysis of the year's cohort to that of previous years (output).

This is a simplification of the overall processes involved, but hopefully this illustrates how an information system takes in data of many different kinds, processes it and then produces information tailored to the needs of the organisation.

Determining the solution requirements

One of the first steps in undertaking the creation of a software solution may well be the determination of what the actual problems are. It will be difficult to begin the planning of the solution if the scope and context of the current problems are not fully understood. What does the solution need to provide? There are a variety of ways in which this can be determined and they all come under the heading of 'data collection'.

Determining the solution requirements is the first activity listed under analysis in the PSM.

Data collection

Organisations are typically motivated to change their information systems in response to problems that may be present. One of the challenges is defining these problems in ways that are quite specific. For example, employees may complain that the system does not perform well during peak times and management may not be able to produce reports in the desired format. The PSM can be used to specifically identify the nature of information problems.

Data collection is an essential part of the analysis phase of the PSM. This is where a systems analyst will collect all of the data on the existing system in order to document its operation and come to a better understanding of the problems that are being encountered.

There are a number of ways in which this can be done. Data can be gathered through the use of interviews, by observing the operation of the information system and by conducting surveys or issuing questionaries.

Before discussing each of these in more detail, it is useful to understand the distinction between quantitative and qualitative data.

Quantitative and qualitative data

All data or information can be classified as being either quantitative or qualitative. Quantitative data is data that can be easily processed in a statistic manner. Usually this means that the data is composed of definite numbers. Qualitative data is data that consists of descriptive details, usually the type gathered in surveys or interviews.

Surveys and questionnaires

Surveys and questionnaires are useful to gather data from employees, clients, management, the community and other stakeholders. This data will often include satisfaction levels with the current system and opinions about how to solve the present difficulties. This form of analysis could also be presented as a test or some form of assessment, if this were appropriate.

Surveys and questionnaires can take a number of different forms. Structured forms such as multiple choice or scaled responses are commonly used as these allow data to be processed easily, but unstructured forms in which opinions can be freely expressed also have their place.

Collecting data in this way can have a number of advantages. As surveys and questionnaires are easy to administer and not expensive, the overall satisfaction with the system can be easily gauged. Despite this, processing responses can be time consuming unless an automated method is used. In addition to this, retrieving completed responses can sometimes be difficult, as those surveyed often feel that there is little incentive to do so. It can

also be difficult to determine who has filled out a survey and therefore how legitimate responses are.

Interviews and focus groups

Interviews and focus groups can be used to obtain qualitative data on the use of the system and attitudes to it. They have the advantage of providing richer data than that obtained through the use of surveys alone. In fact, sometimes interviews and focus groups may shed light on responses that have been retrieved through other means but were not fully understood. Smaller focus groups are preferable to larger ones, as in a smaller group, those participating may be more inclined to respond. Documentation of interviews or focus groups can be tedious, but technologies such as scribe pens, mp3 recorders, web-cams, etc. can make this task easier.

Observations

Observations provide a method by which data can be gathered on how the system operates and how it is used, in an unobtrusive manner. Observations can be conducted in a very structured way with specific actions being recorded using a predetermined method. For example, data could be gathered on how a system operates during a stock-take, with observers recording the interaction between operators and a particular software package. Alternatively, observations may involve an observer simply recording what is occurring in a system as a whole. Typically, a number of individuals are employed to conduct the observations so they can be conducted simultaneously.

Figure 33: Customer survey

Observations can be time consuming and expensive to carry out. Their strength is that they provide a view of the system that is unbiased by the opinions of those within it. It is important to recognise that observations only provide a snapshot of the system within the time frame of the observation. For this reason, the chosen time frame is very important. It is also important to acknowledge that people being observed often behave differently, which may have an effect on the data that is gathered.

Solution requirements

The requirements of a software solution can be classified as either functional or non-functional. Often the difference between functional and non-functional solution requirements is described as being the difference between what a software solution is going to do as opposed to what qualities it has.

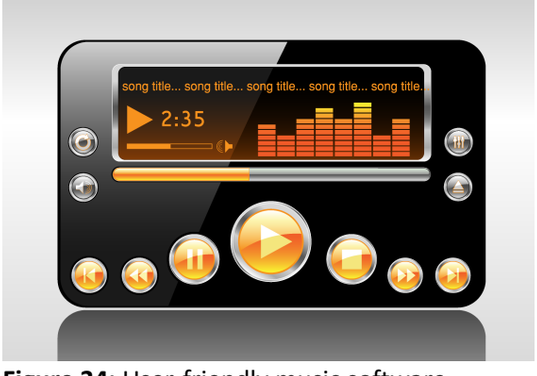


Figure 34: User-friendly music software

Functional solution requirements

Functional requirements are directly related to what the software solution is required to do. What inputs will it receive, what outputs will it generate and how will it behave? The functional requirements of the solution are often determined after an in-depth analysis of what is currently in place and what is required. Much of what we will discuss in the next few

chapters will be concerned with ways in which this analysis can be performed.

Non-functional solution requirements

Non-functional requirements are related to the characteristics of the software solution, such as usability, reliability, portability, robustness and maintainability. Let's look at each of these in turn.

Usability

Usability is related to how easy it is for users to make use of a software solution. It isn't a measure of how happy or content users are with a solution, but rather is measuring how clear, intuitive, logical and accessible a software solution is. A so called user-friendly software solution may work in one environment but fail in another – as the requirement of user-friendliness is strongly dependent on the users themselves. This is an important consideration. It is paramount to have an understanding of the users, their relative skill levels, expertise and needs in order to write a software solution that they would consider user-friendly.

Reliability

Reliability is a measure of how long a software solution can perform its required functions in the operating environment in which it was designed. Often this is expressed as a probability.

Portability

Portability is the ability for a software solution to be moved from one environment to another. In Software Development, portability is generally achieved by separating the logic code from the interface code. If such a separation is performed, a software solution can be migrated to a new environment with changes only being necessary to the interface code (which will save money and time). What's more, the function of the software solution will not have changed from one system to another. If a software solution is being developed for use at an organisation where it will be used across a number of different platforms, dividing the solution up between the logic and the interface will also save a great deal of money and time.

Robustness

Robustness is a measure of how well a software solution responds to poor use or inputs. When designing any software solution, it is very important to validate all of the input to ensure that the software works correctly, but it is equally important to ensure that the program continues to run and will not fail with an incorrect input. A software solution that was not robust, for example, would crash if a user entered some incorrectly typed data into one of its inputs.

Maintainability

Maintainability is mostly concerned with how easily a software solution can be fixed if and when problems occur. In addition to this, the term maintainability is also used to describe how easy it is to make modifications or additions to the software and how well the software copes with changes made to its environment.

There are measures that are used in the field of software development to gauge the maintainability of a software solution over time. The key reason why this is a very important measure is that there is a definite correlation between the maintainability of a software solution and time. Maintainability is generally excellent when a software solution is first implemented, but it decreases as time goes on. As the process of maintaining a solution can be costly, there is often a time when the cost of creating a new software solution is more viable than maintaining an old one. Predicting or detecting when this point has been reached can save an organisation a great deal of money.

Identifying the constraints on the solution

It is very important to understand the constraints that are placed upon a solution. Without an understanding of the constraints, the success of the solution may be in jeopardy.

Economic factors

Probably the most common and obvious constraint is that of cost (economic). This will vary quite a bit from situation to situation and will depend on the size of the organisation, the budget and the scale of the project.

Often when reflecting on the cost of a project, software developers will focus on the resources required, whereas the main cost is generally time. Development time is valuable and it may be that the time frame is set by the production milestones. While this is a factor, the cost of having a software developer or a project team working on a solution over a period of time, may mean that the overall cost of a project is much more expensive that an organization realises. This is especially true when organisations are using their own software developers who are on staff, as they aren't considering what it would cost to employ someone on a contract basis specifically to create the solution. When employing a software developer on a contract basis, the cost of the software developer's time is transparent, and may cause an organization to reconsider their options or their budget.

Technical factors

Security features may be needed or may be part of the existing infrastructure and need to be catered for. Compatibility with existing hardware and software may also be a factor as well as the speed of processing required (or available) and the capacity of the existing system. The solution may need to be developed for multiple platforms as well.

Let's expand upon some of these briefly.

Technical factors: speed of processing

The speed of processing required of a system will vary depending on what the system has been built to do – an air traffic routing system will certainly need to process a large amount of data quickly as opposed to an online diamond purchasing website.

The speed of processing of a system is often exhibited in its response rate. The response rate is a measure of how long it takes for certain actions within the system to be completed. If the response rate is too long, users may get frustrated with the system and will view the software product negatively. Similarly, a low response rate could result in loss of customers and loss of money, as completing a task may take longer than it should.

Speed of processing is often measured in IPS (instructions per second) which in turn becomes MIPS (million instructions per second) and GIPS (giga instructions per second). The number of instructions that a system can process is often not the best way to represent its speed, as the speed can be significantly influenced by factors such as the CPU architecture and memory structure.

Technical factors: capacity

The capacity of the current information system is a significant constraint on any proposed software development. Capacity can include simple factors such as the amount of available hard drive and/or network storage or the amount of memory within the computer systems that will use the software solution. The network infrastructure will place a limit on the amount of data that can be transferred simultaneously. It may also place a functional or physical limit on the number of people that can use the software solution at the same time.

Technical factors: availability of equipment

The availability of equipment may be a constraint in a number of ways. It might be the

case that the software solution is being developed for an information system that is being upgraded and does not have the required hardware and software components as yet. It might also be the case that the current hardware and software will not be upgraded at all to cater for the new solution, so the software developer must work within the reasonable limits of their capacity. This leads into the next technical constraint that needs to be considered – compatibility.

Technical factors: compatibility

As with the previous constraint, a new software solution will often need to work within the bounds of the existing system. This includes being compatible with the existing components. On a base level, a software developer needs to be aware of the operating system that the existing system is utilising. On top of this, a software solution will need to send data to and from related systems and databases and will need to be compatible with the file formats that these systems use. The specifications of the devices within the system will need to be catered for so that the software solution runs in an optimal fashion when being used by these devices.

Technical factors: security

An organisation may have specific requirements in terms of security for the new software solution. Features such as encryption may need to be included and data may need to be stored and accessed from secure network servers within the building or off-site. A software solution may need to be able to be integrated with a SSO (single sign on), which essentially means that a user can access the system once they have logged in to the main portal of the organisation.

Social factors

Constraints on the solution of the 'social' type can include the level of expertise of users, availability of technical support staff, the time available to develop the solution and the availability of equipment. There also may be users in the system who have special needs such as sight or hearing impairment or a disability.

Legal requirements

Without discussing the specifics of the legislation concerning copyright and privacy, it is easy to comprehend how legal requirements within organisations act as a constraint on a software solution. For example, an organisation's privacy requirements may dictate that a software developer needs to protect the data within a software solution and include security features that prevent unauthorised access. This will undoubtedly take time and add to the cost of the development. Chapter 9 explains the relevant legislation in detail and discusses some of the issues surrounding it.

An organisation also needs to consider the ownership of key components. Now that we are in the era of cloud computing, key software solutions may well be located off-site or indeed in another country. It is important that an organisation has a clear agreement with the software solution provider that ensures the ownership of their data remains with them.

Usability factors

Another constraint on the software solution is usefulness. This is not to be confused with the ease of use of a solution.

Usability factors: usefulness

Usefulness is the measure of the ability of something to satisfy need. As a constraint, this may be the requirement from users that a new software solution performs in a particular way or offers a number of core features. This may be a constraint as a direct result of an evaluation of the previous system that may have identified some specific concerns that users have.

Usability factors: ease of use

The ease of use of a proposed software solution is a constraint as it may place some specific requirements on the development of the user interface. There may be aspects of the development that need to concentrate on the function of the new system when used on the types of devices that users have. There may have been difficulties in the past with the number of steps or complexity of tasks within the software. There may have also been issues with the ease with which data was entered and validated.

Identifying the constraints on the solution is the second activity listed under analysis in the PSM.

Determining the scope of the solution

The scope of the solution is similar in ways to the constraints. The factors that may have played a part in forming the constraints of the solution may likewise help to form the scope. In this sense, the scope of the solution may be directly determined by the constraints or the scope (most likely) will fall within the constraints.

The scope defines what the software solution will do and what it won't do. It defines what the boundaries of the solution will be and what benefits there will be for users. In a way the scope identifies the responsibilities and parameters of the solution. Benefits are often stated in terms of efficiency and effectiveness measures.

For example, an organisation might be wishing to develop a software solution for their accounting department. The accounting department, however, might stipulate that the system needs to be ready for the start of the financial year. The original scope of the project may have been too large to fit into this constraint, so the scope of the solution is reduced to ensure that there is enough time to get it finished for the start of the financial year.

It is quite possible that a solution is being developed for just one part of an organisation. In a case such as this, the constraints would be focused just on that one part and would probably not affect the rest of the system (although the rest of the system may have imposed some constraints on the solution). The scope of the solution in this case would be limited to the part of the organisation being examined.

Determining the scope of the solution is the third activity listed under analysis in the PSM.

Project management

The field known as project management is one that can be undertaken by a person within a project as a full-time task. While this is true, having project management skills is essential to the successful delivery of software solutions on time and to specification. The level of project management required is influenced by factors such as the size and complexity of projects being undertaken. For example, there may be a date by which the project needs to be operational or there may be times when the system will be unavailable or the company

Figure 35: An example of a Gantt chart

closed. These would be examples of constraints on the proposed software solution and they may also have an impact on the scope. One of the main tasks that a project manager performs is breaking the project down into a series of key tasks that need to be accomplished. This list of tasks is important for software developers, as it not only helps to structure how the software solution will be put together, but it helps to divide a project up in a team development environment.

Project management enables targets to be defined, resources to be managed, deadlines to be met and changes in circumstances easier to deal with. It is extremely useful in determining budgets and is often used to present a project proposal for tender. Although there are a number of tools that can be used to make this task easier, we will discuss one of the main ones.

Gantt charts

A Gantt chart is a type of bar chart that is used to show how a project's tasks progress from one to another. It was originally created by Henry Gantt, a mechanical engineer, who first published the chart in 1910. A Gantt chart is able to show which tasks can be done simultaneously, and which tasks depend on others to be completed before they can be commenced. These are referred to as dependencies. It also shows times in the project when a certain stage must be reached. These are called milestones. An example of a Gantt chart is shown in Figure 35.

Figure 36: Recording progress against the initial plan

In the Gantt chart shown in Figure 36, you can see milestones at points in the project timeline represented as diamonds. The separate tasks are drawn from their starting date to their finishing date and have arrows leading to tasks for which they are a predecessor (that is, a dependency). The long horizontal bars are used to represent stages of the project as opposed to separate tasks that need to be completed.

Identifying tasks and resources

Time allocation and sequencing are of prime importance in project management. What are the main tasks that need to be completed to take a particular project from start to finish? The identification of these tasks is vital as it will often give those involved in the project a good sense of the scope of what needs to achieved as well as how easily the tasks can be distributed amongst those involved. Not only this, but some tasks will be dependent upon others being complete first (known as a dependency, as mentioned before).

It is important to remember that a project is not just about completing a number of tasks in a set amount of time. It is also about managing resources. Resources can include personnel, hardware, software as well as a variety of other items such as hiring venues for consultations. These resources are only available in finite amounts and so the construction of a project management timeline must account for the shared usage of them.

Planning for things to go wrong is almost as important as planning itself. If sufficient slack is built into the project timeline 'just in case', then unforeseen problems are less likely to cause the project to become late. It is

equally as important to have contingency plans in case a major difficulty occurs. A contingency plan may be one in which the functioning of the old system can be returned for a period of time or a manual system can be put into place so that business operation is not lost.

Using a tool such as a Gantt chart allows all tasks to be visually represented, resources allocated to each task and their progress monitored. This is of prime importance, as project timelines will often change for a variety of reasons as the availability of resources can change, the completion of tasks can take longer than expected (and have a flow on effect to subsequent tasks) or critical deadlines are changed.

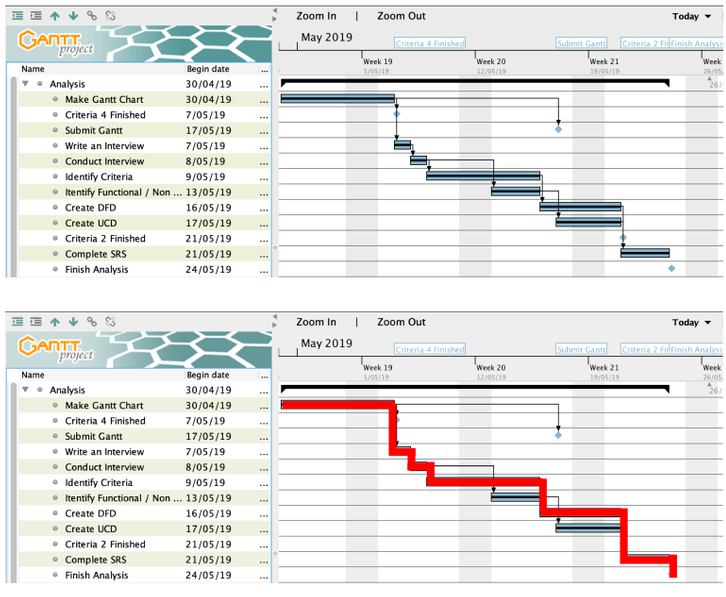


Figure 37: A section of a Gantt chart, and then the same chart with the critical path highlighted.

Recording progress

Recording the progress of a project is almost as important as making the initial plan. Projects almost never progress from start to finish without a variation in times, resources or schedules. While it is important to allow for this as the project plan is being constructed, it is equally important to monitor changes to the project plan as it is being implemented. By keeping a close eye on how the project is progressing compared to the initial plan, it is easier to respond in order to accommodate these changes.

As a project progresses, a good project manager will regularly examine the plan and adjust tasks for the times that they are completed or estimated to be finished. This will in turn have an effect on the overall timeframes within the project and may mean that some resource bookings need to be modified (outside contractors, system testing times).

A good project manager will also make annotations to tasks as changes are made or additions catered for. They will keep a comprehensive log of changes so that when they need to make reports to their superiors or the organisation, they are able to explain the variations to the initial plan that have taken place and the impact of these changes.

Critical Path

Once the initial draft of a project management plan has been created, a project manager can examine the tasks, resources, times allocated and dependencies and gain an overall view of the project. Various milestones will fix the timeline at points in the plan, but other than this, there may still be some flexibility in regards to how long tasks will take and the order in which they will be done.

Of specific interest to a project manager, will be the critical path. The critical path in a project is defined as the longest path from the beginning to the end of the timeline. What this represents is the path along which there can be no delays that do not affect the completion date of the project.

We have just been discussing how the progress of a project will change over time and how important it is to reassess tasks and resources and make adjustments as required. The critical path of a project is important as any changes to the tasks on the critical path will mean that the project will be delayed. A project manager that is aware of this and making frequent adjustments to their timeline, on recording a delay to a task on their critical path, will need to recognize this and make a future adjustment to another task on their critical path to compensate. Tasks that lie on the critical path should have dependencies attached to them and should have no 'slack time' (that is, time at the end of the task in which the task can be delayed without penalty or negative effect).

Development model approaches

While time is in an important factor in determining milestones and the duration of tasks within a project management plan, another important consideration is the development model that will be used.

In the first chapter we introduced the PSM and discussed the four stages within it and their associated activities. The natural way to envision how this would be implemented is in a purely sequential way. That is, starting at the analysis stage, completing the activities in order, and then moving on to the design stage. This linear progression makes sense and is often referred to as the "waterfall model".

In fact, the development models that follow can be applied to any methodology that is being used, and in some cases, will over-ride the stages and activities within it. The PSM lends itself to be implemented using the waterfall model, and this course is set up for it to be used in this way.

The waterfall model

The waterfall model is a development model that moves through the stages of the PSM in order. Each stage is started and the activities within each are completed in order, before moving to the next stage. On first reading of the PSM, it would seem that this is the most logical way for it to be implemented, though in practice, this is not the case. The reason why this model is called the waterfall model, is that once stages or activities are completed, they cannot be revisited. This can be illustrated in this way.

Figure 38 only shows the 4 stages and not the activities, but the concept should be clear. The waterfall model does not allow for any overlap of activities or stages. Once an activity or stage is completed, the model moves on to the next one. Likewise, stages or activities cannot be revisited. Once they have been concluded, they are considered finished.

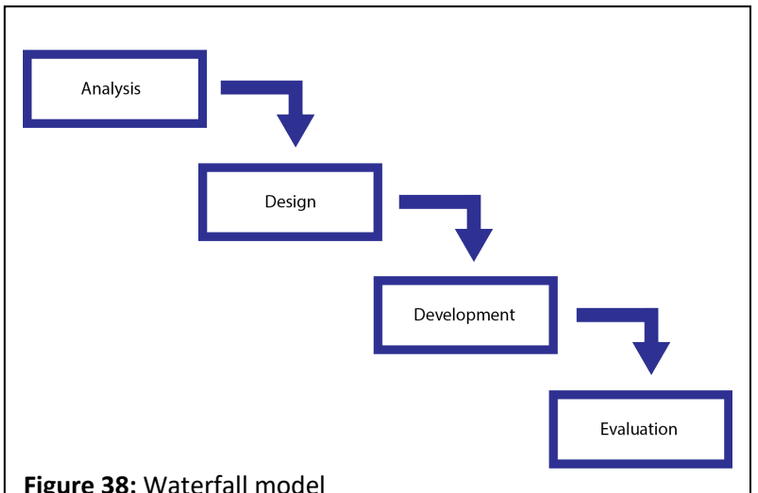


Figure 38: Waterfall model

Advantages:

- Simple, linear progression through all of the stages and activities of the PSM. Easy to understand and use.
- Specific resources and time frames are easy to allocate as the number of steps are known.
- As the stages do not overlap, it is easier to concentrate resources on each task and complete it in the optimal time.

- Easier to keep track or milestones and ensure they are met.
- Works best for projects of a small scale.

Disadvantages:

- Once an application is in the development or evaluation stage, changes cannot be made to the design (if these become apparent).
- Not suited to large scale projects or ones that will take a long time to complete (there is the potential that scope or constraints may change).
- A working prototype or beta version of the application will not be available till late in the development process.

The agile model

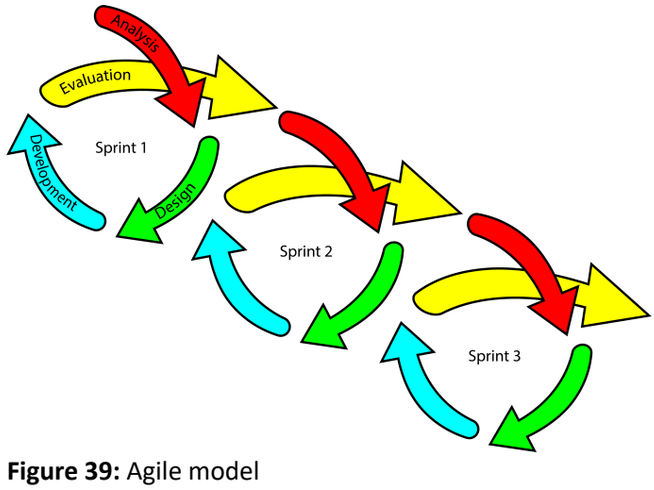


Figure 39: Agile model

Unlike the waterfall model that works methodically from start to finish, the agile model makes a quick start and aims to rapidly perform all of the stages of the PSM (or the methodology being used) over and over gradually improving on the functionality and scope of the final product.

While agile processes are very useful for achieving outcomes that are favourable to all stakeholders, they are prone to causing projects to run over time. For this reason, strict project management is a must. The agile development model can also be susceptible to 'scope creep'. A hallmark of this model is the client interaction. While this is a very positive thing, it does allow the client to add features to the original design or to change the scope once the process is started, which is an advantage of this model but one that needs to be monitored.

A popular way of implementing an agile development model is using Scrum framework. In the Scrum framework, cycles are defined as 'sprints' and are each given a specific (non-moveable) time-frame (typically several weeks). At the start of the sprint, the client meets with the software development team to discuss their needs (which can be specific or general, large or small). All of these needs are listed are developers are assigned to them. Each day, the development team meets (in a scrum) for 15 minutes to discuss their progress. As a team they problem solve impediments, discuss constraints and scope and share their progress. This process is managed by one member of the development team who has been assigned the role of 'scrum master'. The scrum master also ensures that everyone understands the process and the time left in the sprint. At the conclusion of the sprint, the client is shown the progress of the work so far and new goals are set for the next sprint.

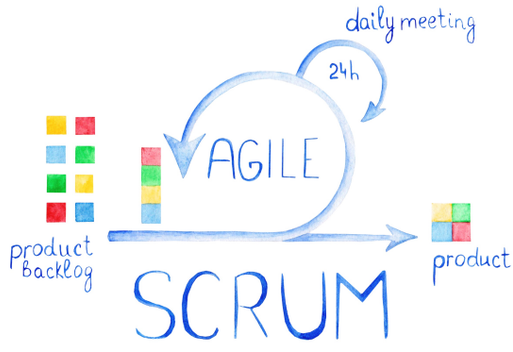


Figure 40: Using a Scrum in the agile model

Figure 41: A Scrum board in action

Advantages:

- There is a quick turnaround from the initial data collection with the client to the delivery of a software product. This promotes discussion between the software developer and the client and leads to changes and enhancements being made to the design for the next iteration.
- Clients are much more included in the development process.
- Agile development is able to adapt to changing circumstances in relation to the design brief.

Disadvantages:

- The scope of the project may not be fully evident at the beginning.
- The project can quickly go off track, especially if the client keeps changing their mind or circumstances keep changing.

The spiral model

The spiral model of development can be seen in ways to be a merger between waterfall and agile. It maintains the iterative style of the agile model, while placing an increased emphasis on risk analysis.

The spiral model has four phases: planning, risk analysis, engineering and evaluation. A software development project repeatedly passes through these phases (which are called spirals in this model). The baseline spiral, starting in the planning phase, requirements are gathered and risk is assessed. Each subsequent spiral builds on the baseline spiral.

Planning phase: Requirements are gathered during the planning phase (that is, an SRS).

Risk Analysis: In the risk analysis phase, a process is undertaken to identify risk and alternate solutions. A prototype is produced in response to this.

Engineering Phase: In this phase software is developed and tested.

Evaluation phase: This phase allows the client to evaluate the output of the project to date before the project continues to the next spiral.

Advantages:

- As spiral is focused on risk analysis, the avoidance of risk is enhanced.

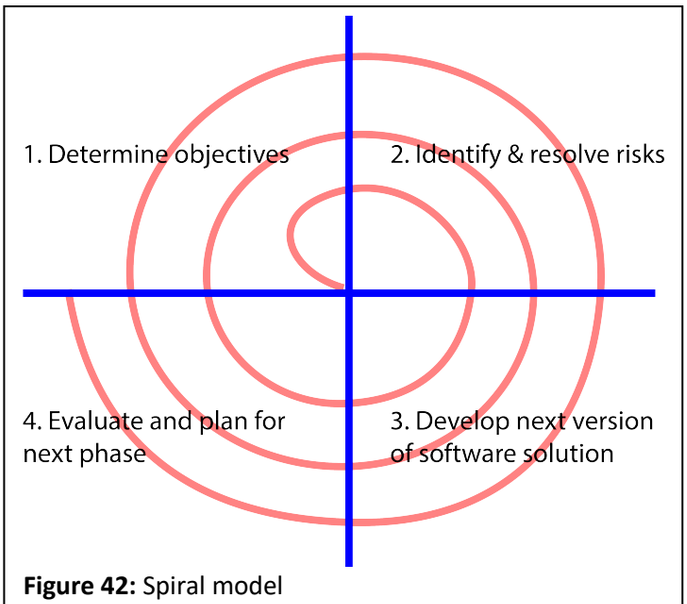


Figure 42: Spiral model

- Good for large projects that have critical elements.
- Additional functionality can be added at a later date.
- Software is produced early in the development cycle.

Disadvantages:

- The risk analysis component is a highly specialised one and requires a software developer that has experience in this area.
- Doesn't work well for small projects.
- Due to the complexity of the phases and the model itself, it can be costly to implement.

Context Questions

1. What should be the first step in planning a project?
2. What is the difference between qualitative and quantitative data?
3. Describe one advantage and one disadvantage of collecting data via observation.
4. Give three examples of constraints that can be placed on a solution.
5. What is the difference between functional and non-functional requirements?
6. What is meant by the term 'robustness' and what can be done to make a solution robust?
7. What is meant by the term 'portability'?
8. What is a focus group and who would be included in such a group?
9. What is the difference between a goal and an objective?
10. What is the purpose of a mission statement?
11. What are the four components of an information system?
12. What does it mean to say that the PSM can be implemented using the agile method?

Applying the Concepts

- See if you can create a Gantt chart to describe the process of building an average house. Divide the project up into tasks and try to visualise the dependencies between them as well as what resources will be needed and when. What tasks would lie on the critical path?
- Try to write a mission statement for your school and then compare what you have written to the actual one.
- Select a number of devices and list as many functional and non-functional requirements as you can for each.
- Imagine you are making a film using the agile method. What would this process look like and what difficulties could you foresee? How would this compare to using the spiral methodology?

Key Skills Checklist

At the conclusion of this chapter, you should be able to address the following key skills. Mark each off as you can achieve them.

List and describe ways in which data can be collected and the benefits of each ☐

Explain the difference between organisational goals and objectives ☐

Explain the difference between organisational goals and objectives and information system goals and objectives ☐

Create a project management plan for a software solution ☐

Determine the critical path in a project management plan ☐

Describe a number of functional and non-functional requirements for a proposed software solution ☐

Discuss the agile, waterfall and spiral development models and their differences ☐

For a proposed software solution, explain the constraints and the scope ☐

Sample Examination Questions

The following sample examination questions can be attempted to test your knowledge of the content of this chapter.

Question 1
Which of the following is not a valid organisational goal?

- **A.** Be profitable
- **B.** Ensure 99% network uptime
- **C.** Manufacture quality products
- **D.** Provide excellent customer service

Question 2
Which of the following is not a valid organisational objective?

- **A.** Reduce customer complaints by 60%
- **B.** Maintain an employee retention rate of 98%
- **C.** Provide care and support for the staff within the organisation
- **D.** Make a profit of $10K per calendar month

Question 3
An agile process is one that:

- **A.** Begins at the start and continues through to the end
- **B.** Can begin and end anywhere in the cycle
- **C.** Is very cost effective
- **D.** Can revisit previous stages to refine the solution further

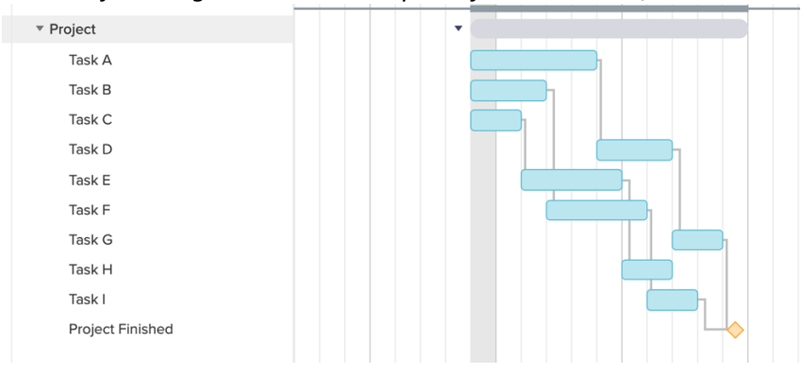
Question 4
A milestone in a Gantt chart is used to show:

- **A.** How many people will be working on a task
- **B.** The resources needed for a task
- **C.** How tasks flow from one to another
- **D.** A point in time when a certain stage must be reached

Question 5
A mission statement is:

- **A.** A lengthy document outlining all business practices
- **B.** Only for public relations purposes
- **C.** A blurb for the website stating name and location and 'about us'
- **D.** A statement of the ultimate goals of the organisation

The following Gantt chart is required for Questions 6, 7 and 8.



Question 6
The critical path in the Gantt chart shown above has a duration of:

- **A.** 8 days
- **B.** 9 days
- **C.** 10 days
- **D.** 7 days

Question 7
If Task G is delayed by 1 whole day, the overall effect on the project will be:

- **A.** There will be no change in the completion date
- **B.** The project will be delayed by a whole day
- **C.** The project will be delayed by two whole days
- **D.** The project will finish one day earlier

Question 8
Task H must wait for both Task E to be completed, before it can begin. In this context, Task E is known as a:

- **A.** Critical task
- **B.** Predecessor
- **C.** Dependent task
- **D.** Resource

Question 9
Paul employs Maxine to create an App for his business. The first thing that Maxine sets about doing is creating a project plan. When she presents this plan to Paul, the first thing that is obvious to him is that she has left a gap between the end of one task and the start of the next one. Explain why she has done this and the effect it may have on the finish date of the project.

_____

_____

_____

2 marks

Question 10
Julie shows some of her proposed solution screen designs to Geoff and asks for his feedback. In doing so, she mentions to him that he needs to ensure that his feedback is accurate as she does not believe in agile development.

a. What is agile development?

_____

_____

1 mark

b. Discuss the differences between agile and waterfall development.

_____

_____

_____

_____

4 marks

Question 11
What is a mission statement?

_____

_____

1 mark

Question 12
Geoff is in charge of a large software implementation in his workplace, and has constructed a detailed Gantt chart to manage the process. Two weeks into the implementation, Geoff's colleague Regina has noticed that he has not updated the chart. When she questions him about this, he states that he has no intention of modifying the chart at all. List three problems that could arise as a result of this.

Problem 1: _____

_____

Problem 2: _____

_____

Problem 3: _____

_____

3 marks

Question 13
WeCare is a support service for those that need urgent maintenance work to be done in and around their home or residence, but cannot afford to pay for it. Many of the clients are either elderly or out of work. Write two goals that would be suitable for the WeCare information system.

Goal 1: _____

Goal 2: _____

2 marks

Sample Examination Answers

Question 1
Answer: B

Organisational goals are, by definition, not quantitative in nature.

Question 2
Answer: C

Objectives always have measurables associated with them.

Question 3
Answer: D

Question 4
Answer: D

Question 5
Answer: D

Question 6
Answer: B

Longest path from beginning to end.

Question 7
Answer: B

Task G is on the critical path, therefore a delay of 1 day will result in a delay of 1 day for the entire project.

Question 8
Answer: B

Question 9
The reason Maxine has left a gap between the end of one task and the start of another is to allow for any unforeseen events or delays in the project. By doing this, the chances that delays during stages of the project will also delay the finish date of the project, are minimised.

Question 10

a. Agile development is the process by which stages in the PSM are cycled – beginning and ending with consultation and feedback with the client.

b. The opposite is Waterfall development – in which the stages are done one after another with no repeating or cycling back. While waterfall development is quicker than agile development, it also does not respond to feedback (or give the client as many opportunities to provide this). Agile development is more thorough, but takes longer (and therefore costs more). Clients tend to be much happier with agile development as they are kept 'in the loop' much more.

The question stem 'discuss' means that you should explain the pros and cons of each of the things that have been mentioned. Another clue to the response that the examiners are looking for is the number of marks. As this is a 4 mark question, it is important to ensure that you list 4 distinct points.

Question 11

A mission statement is a statement about the goals of the organisation and why it exists.

Remember that a mission statement is often very broad and while it is important, it does not go into specific detail about objectives.

Question 12

a. Milestones may not be met.

b. If the project is not running exactly to time, it may mean that resources that are booked for key parts of the implementation will not be available (given that they will be required later or earlier).

c. Parts of the implementation may begin (at their allotted time) without the pre-requisite tasks they need to be completed having been finished. This may lead to confusion and wasted time.

Question 13

Goal 1: Record the jobs that clients register with WeCare.
Goal 2: Keep track of the progress of jobs.

Other goals could be equally valid involving communicating with trades-people or maintaining the register of clients.

Chapter 5
Analysis tools

The chapter covers Unit 3: Area of Study 2 key knowledge:

- *Data and information*
  - *KK2.6 Features and purposes of software requirement specifications*
  - *KK2.7 Tools and techniques for depicting the interfaces between solutions, users and networks, including use case diagrams created using UML*
  - *KK2.8 features of context diagrams and data flow diagrams*

Key terms: Unified Modelling Language (UML), use case diagram, actor, use case, system boundary, association / communication, includes, extends, Data Flow Diagram (DFD), entity, data flow, data store, process, DFD level, context diagram, Software Requirements Specification (SRS).

Study roadmap

	Unit 3		Unit 4	
Ch	Area of study 1 – Programming	Area of study 2 – Analysis and Design	Area of study 1 - Development and Evaluation	Area of study 2 - Cybersecurity: software security
1	✓			
2	✓			
3	✓			
4		✓		
5		✓		
6				
7				
8				
9				
10				

Analysing an information problem

There are a large number of tools that can be used to analyse an information problem in an organisation. The use of these tools makes it easier to identify the interfaces between solutions, the users of the system and the network within which the solution needs to operate. The first of these tools is called a use case diagram (UCD) and these can be graphically represented using Unified Modelling Language (UML).

Unified Modelling Language (UML)

Unified Modelling Language (UML) is a standardised modelling language that encompasses a large variety of diagram types that can be used to represent many aspects of software development. Originally conceived in the mid 90s, UML had a major revision in 2005

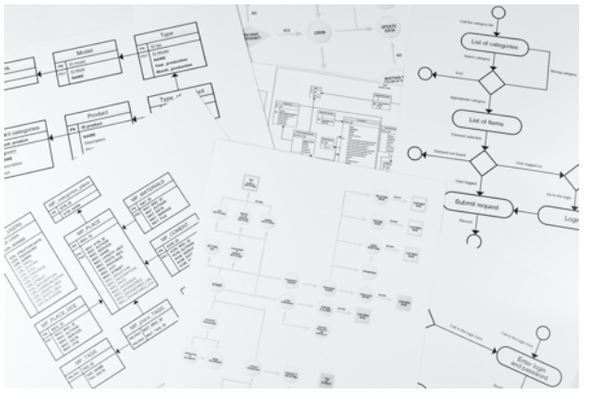


Figure 43: A selection of diagram types from within UML

and the current standard is 'UML 2'. The important aspect of UML is that each diagram type is represented and utilised in a strictly defined way allowing references to be easily created and diagrams easily compared.

Use Case Diagrams (UCDs)

A use case diagram is essentially a structured 'story' depicting the functional aspects of a system within an organisation. It is used to depict the goals of the system and how people and organisations interact with the system to achieve these goals. It can be helpful to imagine a use case diagram as depicting workflow, but it is much more than this. It can be used at a number of stages in the PSM – for example, it can be used as a training aid once a software solution has already been created. In the context of this chapter, we will discuss how use case diagrams are used in analysis.

A use case diagram has a number of elements. Let's look at what these are.

Use case elements

Actor

An actor is the term used to denote a role that is external to the system. They are usually stated as a noun. The actor is often a person or organisation. Stick figures are often used to denote actors, but in the case of organisations this can be misleading – thus boxes can also be used. Actors connect to use cases.

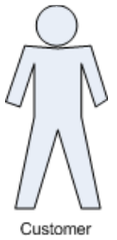


Figure 44: An actor

Use Case

A use case is a representation of a function within the system generally beginning with a verb. A use case is shown as an ellipse with the function written inside it. As use case diagrams are often the first tool that is used in coming to an understanding of how a system works, each use case is often quite broad.

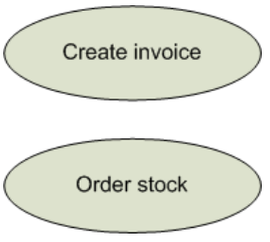


Figure 45: Use case

System Boundary

A system boundary is a rectangle that is drawn around all of the use case symbols and represents the confines of the system. The actors are outside of the system boundary. Although there will typically be only one system boundary, it is possible to have sub-systems represented by additional system boundaries inside of each other.

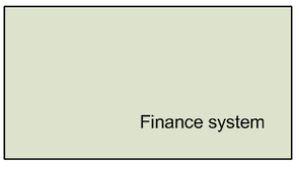


Figure 46: System boundary

Associations / Communications

Lines showing the links between a use case and an actor(s). A use case can be carried out by many actors and an actor may carry out many use cases.

Figure 47: Association

Includes

Dotted lines, with arrowheads and the text "<<includes>>" showing the links between use cases. Usually it indicates that the functionality of a use case is used in another use case.

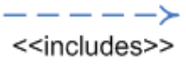


Figure 48: Includes

Extends

Dotted lines, with arrowheads and the text "<<extends>>" showing that the functionality of a use case contributes to (or enhances) the functionality of another use case. This can be conditional on an actor being a member of a particular group. For example, "{is administrator}".

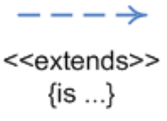


Figure 49: Extends

Use case diagram examples

Student attendance system

The first step in the construction of a use case diagram is determining how many actors are interacting with the system. For the student attendance system being analysed in this example, there are four actors: administrators, teachers, students and parents. Figure 50 on the next page shows the completed UCD.

The next stage is to draw the system boundary. In this example, the system boundary is the student attendance system.

Next we draw the use cases themselves within the system boundary. These are the broad functional aspects of the system. Connections are drawn between the use cases and those actors that either supply data or retrieve information in some way. The description of each use case should explain what sort of transfer of data or information is occurring.

Notice that in some cases, the connections can be drawn as arrows instead of as plain lines (although the example shown doesn't have any connections like this). Arrows indicate when an actor has initiated the use case and is an optional element that can be included. Not all use cases can feature a connection such as this but they do convey extra information that is quite useful when interpreting the diagram.

A use case diagram provides a simple and easy to read representation of the functional aspects of a system. It is by definition quite broad. Use case diagrams are not intended to be the only tool used in analysing a system and the danger is that if they are used in this way, they can become bloated and difficult to interpret. They are best used as a starting point before using a more detailed analysis tools like data flow diagrams.

One of the main criticisms of Use case diagrams is that they can only be used to represent certain aspects of the way a system operates (or is designed to operate). It does not represent algorithmic processes well at all and instead provides an overview. In addition to this, there is no standard that is fully agreed upon as the Use case diagram standard. For this reason, it is important to be consistent in the diagrams that you are creating.

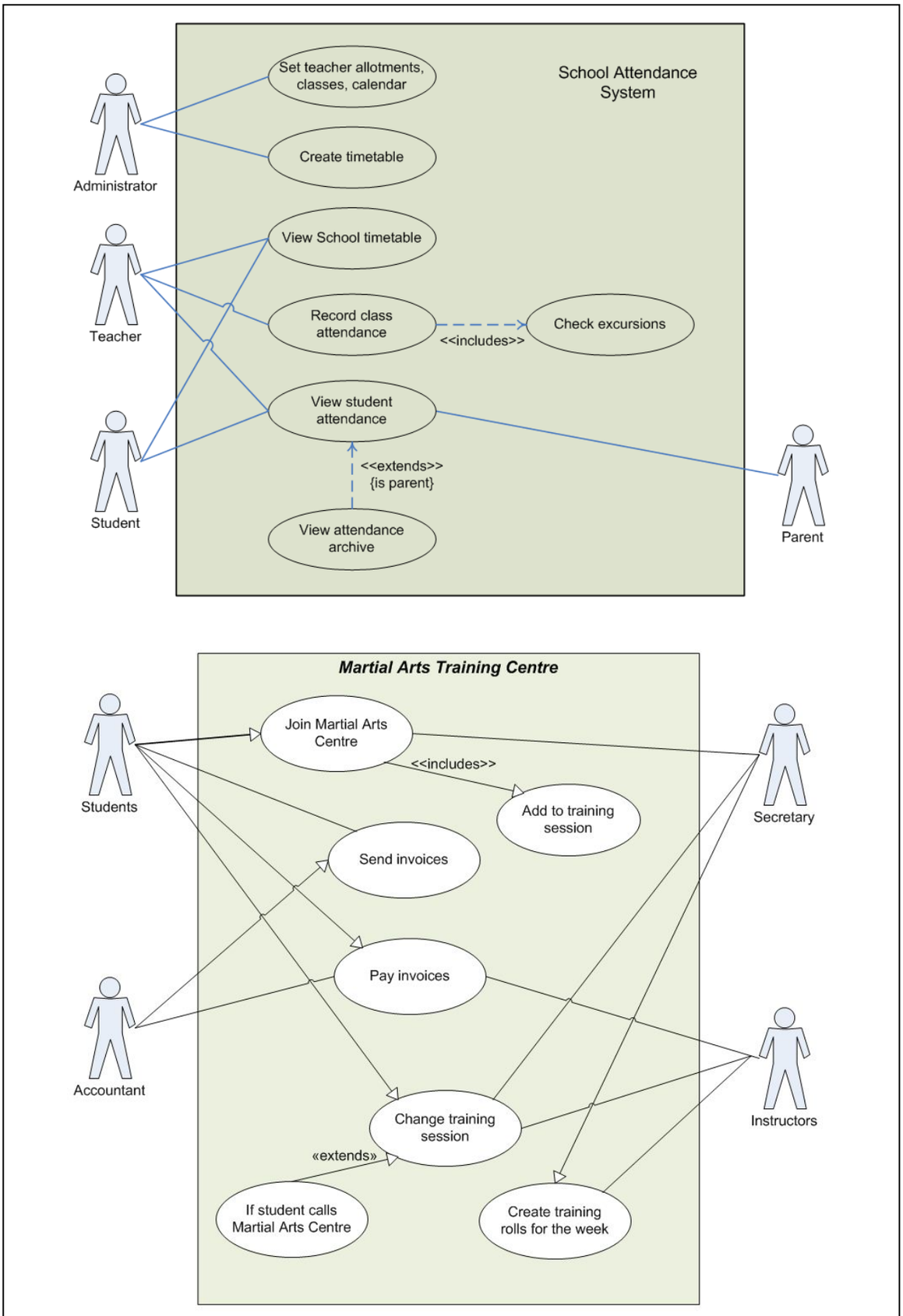


Figure 50: Use Case Diagram examples

Martial arts training centre

The Martial Arts Training Centre example also shown in Figure 50, is one that consists of four actors: students, instructors, the secretary and the accountant. Students contact the centre to join and this includes being added to a training session. The accountant generates invoices and sends them to students who arrange for them to be paid. The secretary creates rolls of all of the students in each training session, and these are given to the instructors. Students often change their session times or notify the centre when they will not be attending. This is either done in person or by phone.

Data Flow Diagrams (DFDs)

Data flow diagrams (DFDs) provide a method by which the movement of data through a system can be visually represented. In particular, a DFD makes it easy to track the progress of data as it is input into a system, the ways in which it is processed, what information is produced, how it is stored and how the information is ultimately used.

The first step in learning and understanding how to create DFDs involves becoming familiar with the symbols that are used and the rules that govern their use. The following section describes the symbols that are used in DFDs and also describes what they contain and the way they should link to other symbols.

DFD Symbols

Entity

An entity is a person, agent or company outside the organisation being examined, that provides some data to the system or receives some information from it and is normally labelled with a noun. It is important to understand that an entity does not process information in any way. Note also that an entity symbol should only be used for those people or organisations that are outside of the organisation. In other words, anyone working for the organisation should not be represented as an entity.

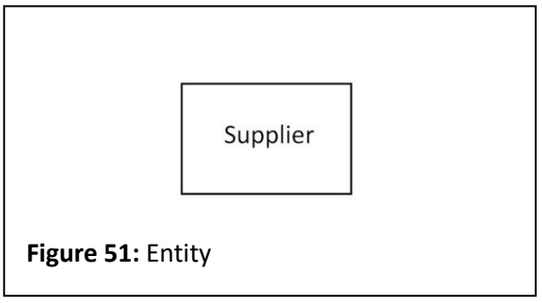


Figure 51: Entity

Data flow

A data flow is represented by an arrow from one symbol to another and generally begins with a verb. There are some invalid combinations of symbols which can link the two ends of the data flow, and these are discussed in the next section. The data that is being transferred should be written on top of (or next to) the data flow. This data could be specific (e.g. ID number, surname) or it could be general (e.g. order details).

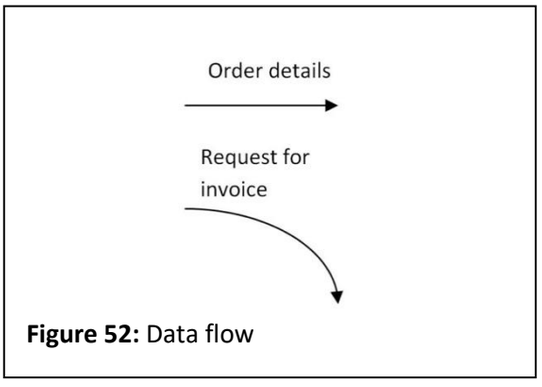


Figure 52: Data flow

Data store

A data store symbol represents a storage location within the organisation. It is labelled with the name of the file or the location. A data store only holds data and does not process that data. It is also possible to represent manual data stores such as filing systems or timesheets in this way.

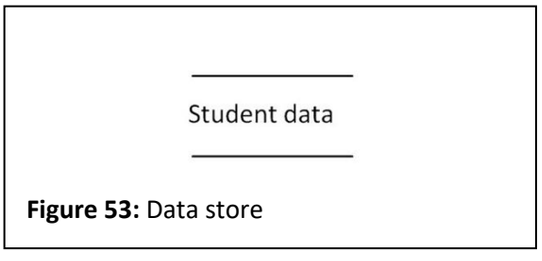


Figure 53: Data store

Process

A process symbol is used to describe any activity that transforms data and generally begins with a verb. The activity could be performed by someone within the organisation, or it could be an automated one. Processes are numbered in the order in which they occur, and contain a description of the task in question.

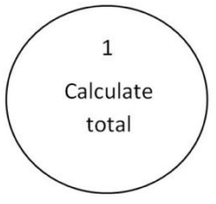


Figure 54: Process

DFD Rules

There are a number of rules that need to be followed in constructing a DFD. Some of them are fairly logical while others have been formed to ensure that diagrams are neat and remain easy to interpret.

General format: left to right, top to bottom

As a general convention, creators of DFDs try to have the DFD flow from the top left corner of the page to the bottom right (in a clockwise direction). This is not always easy to accomplish, but aiming to have a DFD constructed in this way aids readability.

Data cannot flow between these symbols: *Entity – Entity*

An interaction between two entities is outside of the organisation and is therefore not of interest. The only time that it is going to be of interest to the organisation, is if the organisation is facilitating this transfer of data in some way. If this is the case, a process needs to be included between the two entities to describe what is occurring.

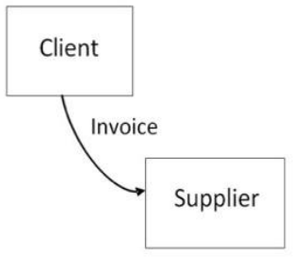


Figure 55: Entity-entity data flow

Data cannot flow between these symbols: *Data Store – Data Store*

If a data store is connected to another data store like this, there is no description of the process that is occurring. For this sort of transfer data to exist, a process needs to be placed in between the two data stores to explain what is happening.

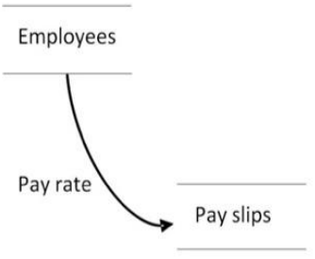


Figure 56: Data Store – Data Store data flow

Data cannot flow between these symbols: *Data Store – Entity* or *Entity – Data Store*

For the same reason that data cannot be passed from data store to data store, data cannot be passed directly between an entity and a data store. For this to be valid there needs to be a description of the process that is being carried out. This is especially true considering that an entity by definition exists outside of the organisation – and it would be highly unusual for them to be writing data or retrieving data directly from within the organization.

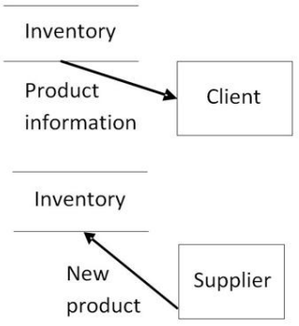


Figure 57: Data Store – Entity data flows

Avoid 'black holes'

A black hole is a term that is used to describe the situation that occurs when data flows into a process but nothing is produced. If the process is doing something to the data and there is no data flow exiting the process, then why is the process necessary at all?

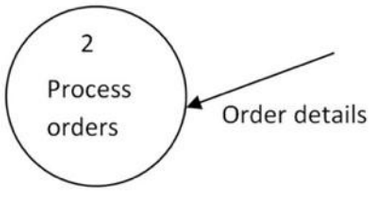


Figure 58: Avoid 'black holes'

Avoid 'magic processes' or processes that do not transform data

Be very careful that the data flows that are going into a process are enough to produce the required information. The process should not be 'creating' information out of nothing.

Another situation to avoid is one in which a data flow enters and leaves a process unchanged. In such a case, there is no indication of what processing has been done (if any).

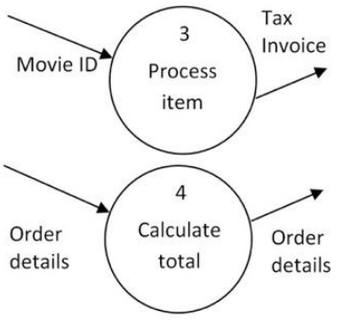


Figure 59: Avoid 'magic processes'

Data that is rejected or discarded is represented by a data flow to nowhere

Data that is rejected or discarded from a process can be an important part of what is taking place, even though the data may not be used for anything else. Often data that does not pass validation will fall into this category, as will data that is redundant or not needed.

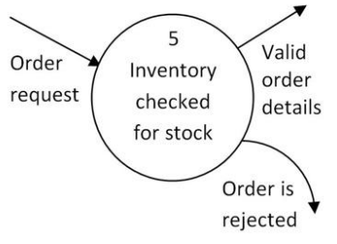


Figure 60: Rejecting data

Ideal DFD size is no more than eight processes

This is more of an opinion than a set convention, but it is generally understood that the size of a DFD is best kept to a manageable level of detail. If it is not possible to break a problem down into eight processes or fewer, then the way that the problem is being divided up needs to be re-examined. There is a mechanism by which DFDs can be further broken down and this will be discussed after the detailed DFD example that follows.

Process descriptions should be meaningful

When constructing a DFD, it is important to ensure that the descriptions that are placed inside processes are meaningful. It can be a good practice to try to include a verb, as the nature of a process is that some task is being carried out. For example, using words like: 'calculate', 'validate', 'add', 'remove', 'edit', 'archive', 'process' and 'compile' is highly desirable.

A worked example: River's Cafe

Let's have a closer look at how DFDs work by constructing one step by step. Consider the description of the process of placing an order at a cafe.

The current system of ordering at River's Café works as follows. When a patron arrives at the café, they place their order with one of the waiting staff who writes it down on a note pad. This order is entered into the computer system by the member of staff currently on the register. The order is printed out and taken to the kitchen where it is placed in the order queue. The kitchen prepares orders based on the orders that are in the order queue and discards them when done. The customer is presented with a bill which they pay before they leave. At the end of the day, all of the orders are tallied and stock ordered from the supplier as needed.

The first step in constructing a DFD should be working out what the separate processes are. Read through the case study and try to pick out the main processes that are taking place.

Processes – River's Café

1. Customer places order
2. Order entered into system
3. Kitchen processes order
4. Bill produced
5. Order is placed with supplier

These are the main processes that are taking place. It is possible to pick out more processes than this or to phrase these in a different way. The important task is to represent the data flows within the system as these are dependent on the number and types of processes that you identify, there is more than one way to do this correctly. We will firstly draw our processes in a roughly clockwise fashion on the page, as shown in Figure 61.

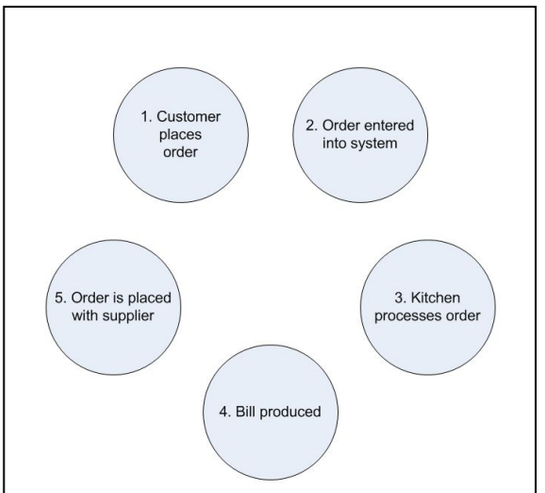


Figure 61: River's Café DFD example

There are three data stores and two entities that are obvious to us on first reading, so let's add them into our diagram. Our new DFD is shown in Figure 62 below.

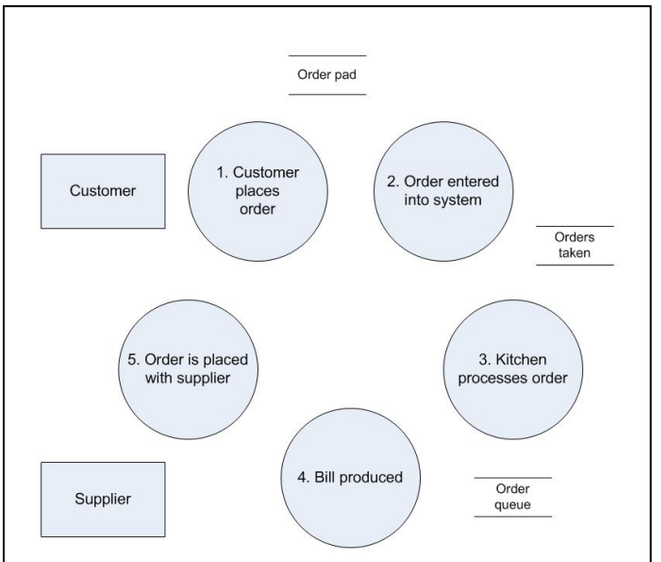


Figure 62: River's Café DFD example continued

Let's now start working through each of the processes and turning them into data flows. Note that when drawing a DFD in this way, it is often necessary to move objects around to make space for data flows or to position them in more convenient locations. You will notice this occurring in this example. Rather than reformatting the diagram so that everything is in perfect position from the start, I have left the original diagrams the way they were laid out from the start. This way, the example also demonstrates how diagrams can evolve in this fashion.

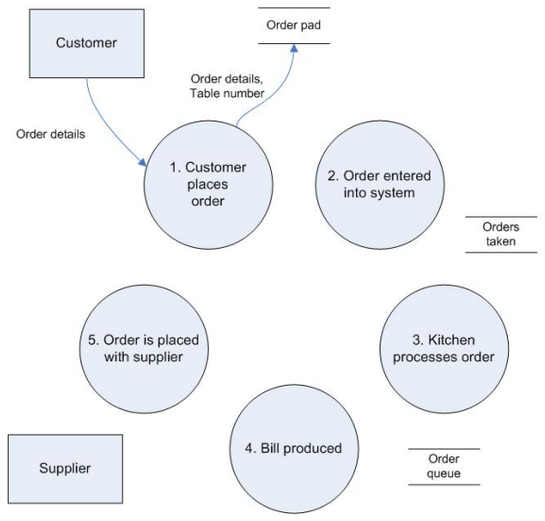


Figure 63: River's Café DFD example continued

1. Customer places order

When a customer places an order, they give their choices to one of the waiting staff who writes it down on an order pad. Any record of data or information constitutes a data store. In this case, the data store is a physical note pad. The member of the waiting staff would also take note of the table number while taking the order.

Note that there are many more exchanges of information taking place in this process. The customer would probably ask what specials are on offer for the day and the member of the waiting staff may or may not need to check this with the kitchen. It may also be the case that some choices are not available or the customer has specific questions about how dishes are prepared. At this 'level' of DFD, we are deliberately keeping the number of processes to a minimum and the data flows simple. More complex DFDs can be constructed using the process known as 'levelling' which is described at the end of this example.

It is important at this stage to only represent what is documented as taking place. It is very easy to guess that some data flows would occur and fill them in, but if this is done then it is hard to analyse the current processes for deficiencies.

2. Order entered into system

When the order is entered into the system, the data is processed in a number of ways. The fact that the data is entered implies some validation takes place although how much the data is validated and in what ways is not known (and could well be a problem with the current system). The order itself is printed out (in a specific format) so that it can be physically placed in the kitchen order queue. As well as this, the details of the order are saved so that

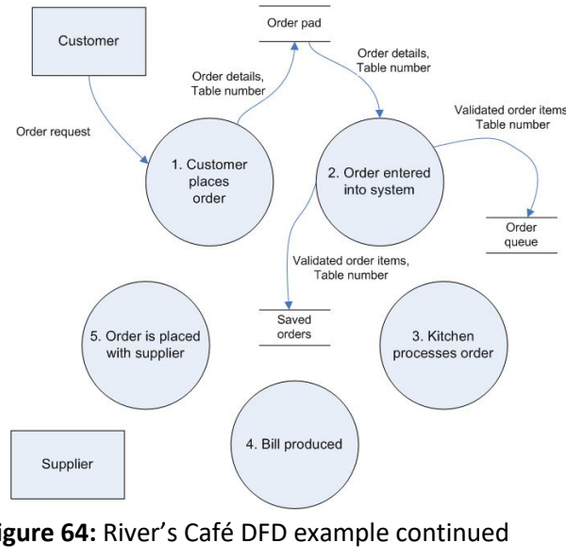


Figure 64: River's Café DFD example continued

later on the amount of food/produce that needs to be ordered can be calculated.

Note that the presence of the 'saved orders' data store was not obvious at the beginning, but has presented itself as being required by this process and subsequent ones.

3. Kitchen processes order

When the kitchen is ready to process an order, the next order is retrieved from the order queue. Once completed, the order is discarded.

These data flows are shown in Figure 65 below.

4. Bill produced

When the customer is ready to receive the bill, they notify the waiting staff who retrieve the order details from the computer and calculate what is owing. This is then formatted into a bill which is given to the customer.

Note that the order in which the data flows occur is not represented in a DFD. We know that the customer requests the bill and that the bill is produced and then given to the customer, but this order is not obvious by looking at the data flows themselves.

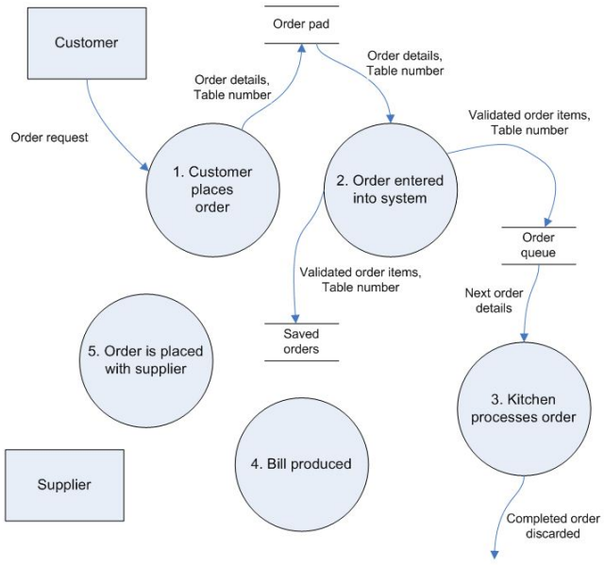


Figure 65: River's Café DFD example continued

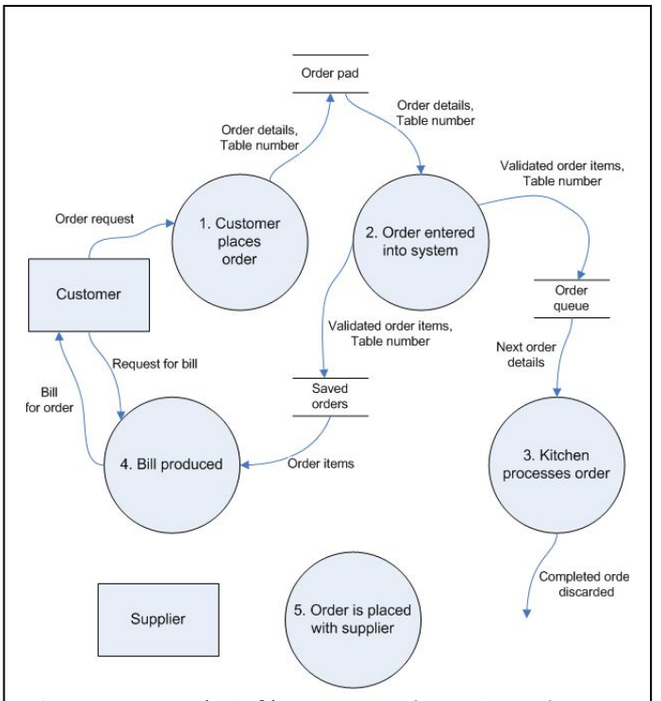


Figure 66: River's Café DFD example continued

5. Order is placed with supplier

The orders for the day are retrieved and analysed to determine what needs to be ordered. This order is then sent to the supplier (or suppliers).

Now that the DFD is complete, you will be able to get a better picture of what is going on at River's Café. In fact, as you worked through this example one step at a time, some questions may have occurred to you. For example, if the customer changed their mind after they had placed their order, what implications would this have for the order in the kitchen order queue as well as their bill? If the customer doesn't request the bill at all, does this mean that the bill is not generated?

This is one of the advantages of creating a DFD to analyse the data flows within an organisation. Not only can the finished product give you a picture of how things work, but creating the DFD causes you to

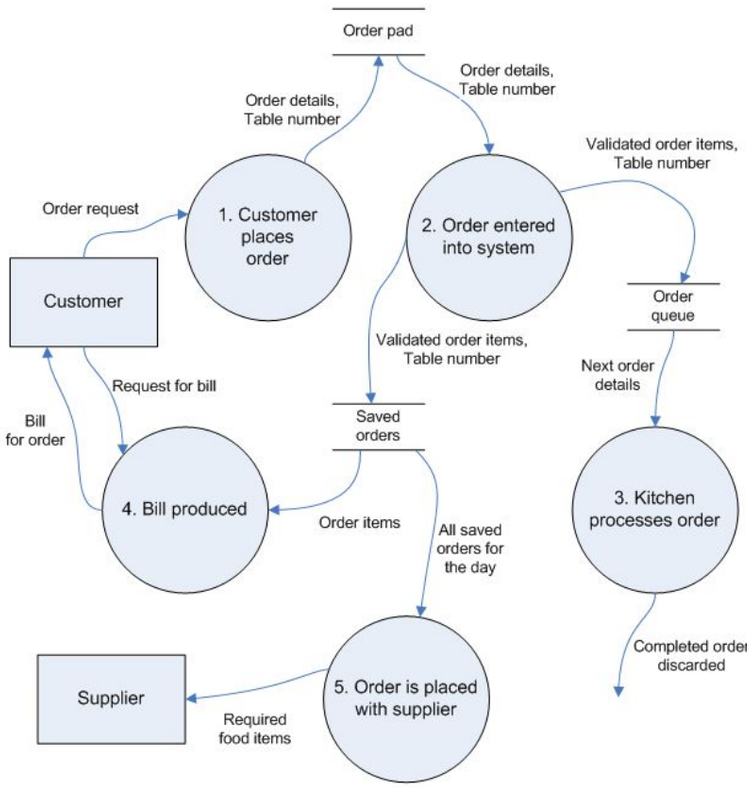


Figure 67: Completed River's Café DFD example

that level would be called a Level 2 DFD. Likewise a process in the Level 2 DFD could be chosen to expand to another level, and so on.

By convention, the processes in a Level 1 DFD are numbered starting from '1'. If the first process was chosen to expand, the processes would be numbered '1.1', '1.2', '1.3' and so on, as shown in Figure 68 below.

think about how processes are achieved in detail.

Levels of DFDs

The DFDs that we have been examining are also called Level 1 DFDs. They form the starting point from which processes can be selected and broken down into more detail. By design, a Level 1 DFD is fairly broad. Once you have created this first DFD, you can select any process and create a whole new DFD that describes how that process is accomplished. This process (known as levelling) could be done almost indefinitely. Each subsequent DFD is named with a new level, so the first time a process is used as the basis for a whole new DFD, it is called a 'Level x DFD'.

For example, if process number '3' was chosen to expand into another level,

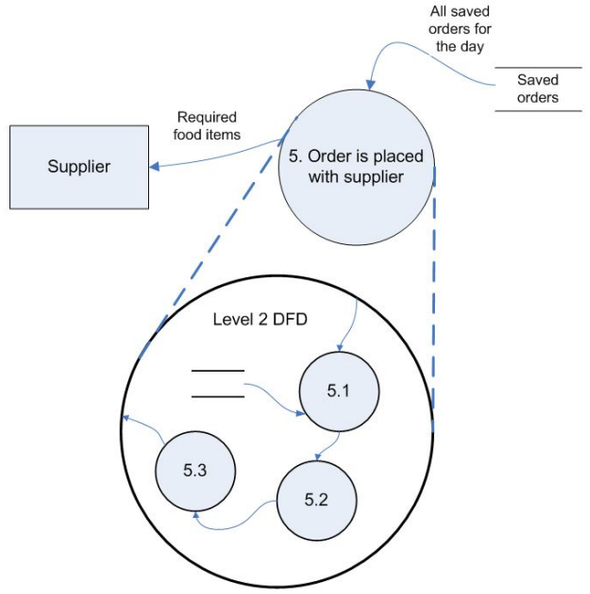


Figure 68: River's Café DFD example continued

Context diagrams

A context diagram is a special type of DFD. It is in many ways a simpler version of the level 1 DFD in that it does not show any of the internal processes inside the organisation. A context diagram focuses on the interaction between the organisation's information system and the external entities that supply data or receive information.

For example, the previous DFD example which looked at processes within River's Cafe would have a context diagram that looked like the one in Figure 69 below.

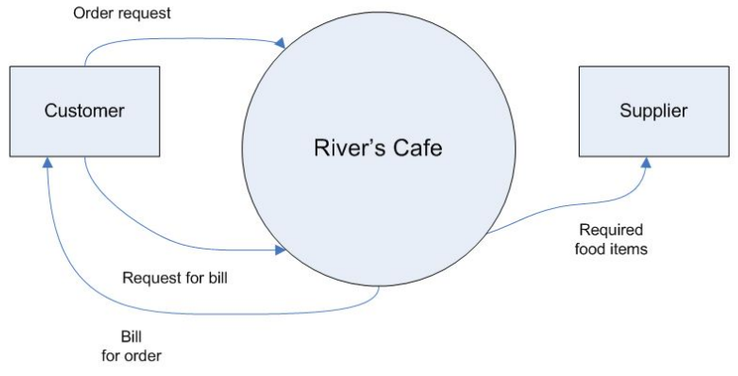


Figure 69: River's Café context diagram

Note that this context diagram does not contain any reference to the internal processes within River's Cafe. In the same way that a DFD can be expanded upon by taking the central part of a context diagram and expanding it, we end up with a level 1 DFD. Because of its relationship to a DFD, context diagrams are often referred to as 'Level 0 DFDs'.

You will also have noticed that the waiting staff are not included in the DFD or in the context diagram. This is because DFDs focus on the processes within an organisation and not on individuals. The waiting staff are involved in carrying out many of the processes shown, but the main purpose of creating a diagram like this is in the understanding of what is taking place. It may be that staffing will change in an organisation from time to time, but the processes will still be carried out. A better diagram to represent the role that the waiting staff play in the organisation, would be a use case diagram.

Software Requirements Specifications (SRS)

All of the features of a software solution that have been discussed so far – timelines, resources, functional and non-functional requirements, constraints and scope are documented in the form of a Software Requirements Specification (SRS).

An SRS is a complete description of what the software solution will need to do. It has the purpose of bringing together all of the data that has been gathered and all of the analysis that has been performed, so that software developers can refer to this information in one document. The content of an SRS can vary depending on the scope of the project and what is felt to be important in the particular situation. Often tools such as Use Case Diagrams and Data Flow Diagrams / Context Diagrams are included as they can show diagrammatically how users will interact with the system and the ways in which data will be transformed into information.

The key elements of an SRS are commonly:

1. An introduction.

   The introduction outlines the purpose of the software solution, its constraints and its scope.

2. A description of the proposed software solution.

   This includes what functions it will perform, what characteristics the user interface will have and any dependencies.

3. The specific requirements of the software solution.

   This section can be the most detailed, often with separate sub-sections detailing the user interface as well as the functional and non-functional requirements.

4. A description of the environment within which the solution will operate.

SRS documents vary in size and content. Complex software development projects will require a significant investment in time to ensure that the SRS is completed and contains all of the necessary elements.

The goal of an SRS is to bring together all of the proposed elements of a software development into one document. This document is then able to be handed to the client, discussed, amended and then ultimately signed off, so that design and development can begin. The investment in time in producing an SRS ensures that the software developer and the client are on the same page and have a common understanding as to the scope of the project. It provides a software developer with a concrete roadmap that has been agreed upon by all parties.

Software requirements specification (SRS): The intended purpose and environment of a software solution. It documents the key activities associated with the analysis stage of the problem-solving methodology. Features of an SRS should include a description of the functional and non-functional requirements, system and technical requirements, constraints, scope and assumptions..

VCAA VCE IT Study Design 2020-2023 Glossary

Context Questions

1. What is UML?
2. What is a use case diagram?
3. Give two examples of possible actors in a use case diagram?
4. In a use case diagram, what does a system boundary represent?
5. In what ways is a DFD different from a use case diagram?
6. Give examples of three physical objects that could be data stores in a DFD?
7. Why is it convention not to represent data flows between entities in a DFD?
8. Why is it a problem when the data flows into a process are the same as the data flows out of a process?
9. What is the process known as DFD levelling?
10. What are the main advantages of a context diagram over a DFD?
11. What is a Software Requirements Specification used for?

Applying the Concepts

- Draw a use case diagram to describe the process of approving an absence (once a student has returned to school after being sick).

Key Skills Checklist

At the conclusion of this chapter, you should be able to address the following key skills. Mark each off as you can achieve them.

Draw a use case diagram to represent the interactions in an information system ☐

Read through a documented process and identify the entities, processes and data stores ☐

Draw a Level 1 DFD to represent a documented process ☐

Draw a context diagram from a supplied Level 1 DFD ☐

Gather the required components to create a Software Requirements Specification ☐

Sample Examination Questions

The following sample examination questions can be attempted to test your knowledge of the content of this chapter.

Question 1
The term UML stands for:

- **A.** Unified Modelling Link
- **B.** Unified Modem Linkage
- **C.** Unlinked Mode Linkage
- **D.** Unified Modelling Language

Question 2
The term UCD stands for:

- **A.** Use Case Diagram
- **B.** Used Cases Diagrammatic
- **C.** Use Case Drawing
- **D.** Unlinked Carrier Diagram

Question 3
In a DFD, an entity represents:

- **A.** A person inside the organisation that deals with data
- **B.** A system within the organisation that processes data
- **C.** A system outside the organisation that processes data
- **D.** A person or company outside the organisation that receives or provides data

Question 4
Which of the following is **not** an element in a Use Case Diagram?

- **A.** Actor
- **B.** System Boundary
- **C.** Includes
- **D.** Data Flow

Question 5
An element of a DFD which transforms the data coming into it is known as a:

- **A.** Data flow
- **B.** Data store
- **C.** Process
- **D.** Entity

Question 6
Which of the following should not be used as an actor in a Use Case diagram?

- **A.** Maxwell Smith
- **B.** System Administrator
- **C.** Accounting Department
- **D.** Time sheet software

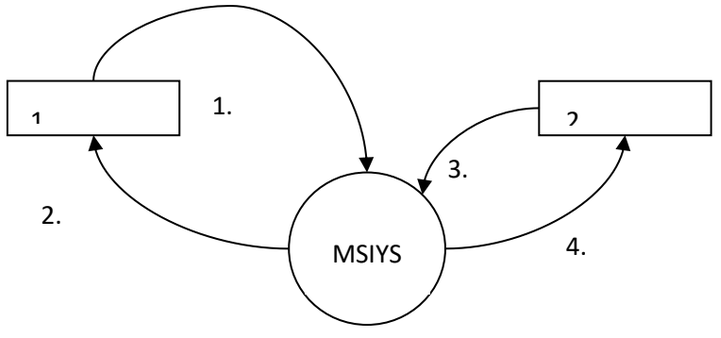
Question 7
Each of the items described below have been taken from a DFD. State whether the description involves a **process** or a **data store**.

Item Description	Process or Data Store
Combine this year's financial figures with those of previous years.	
Retrieve the number of items from the inventory database.	
Archive the month's data for future reference.	
Discard the daily calculations.	
Calculate the profit margin for each stock item.	

5 marks

Question 8
Phillip is the manager of a collaborate work space called 'My Space Is Your Space' (MSIYS). He receives many requests from clients in relation to catering. Phillip would like to set up the MSIYS Café to allow clients to order via an App for table service or local Ooba Eats establishments. It would work in the following way.

When placing an order, the user of the App would scan a QR code (which would be located on every desk and meeting room within the office area), so that the café will know where to bring the order. The App will have a list of café items that can be ordered (some of which may vary from day to day). The App will also be able to display menu items from local cafes and restaurants, that the App will then place orders with, via an Ooba Eats API (a code module that will be supplied by Ooba Eats that will integrate with the App seamlessly). Phillip would like to create a context diagram to represent this process. Label the entities and data flows in the diagram below:

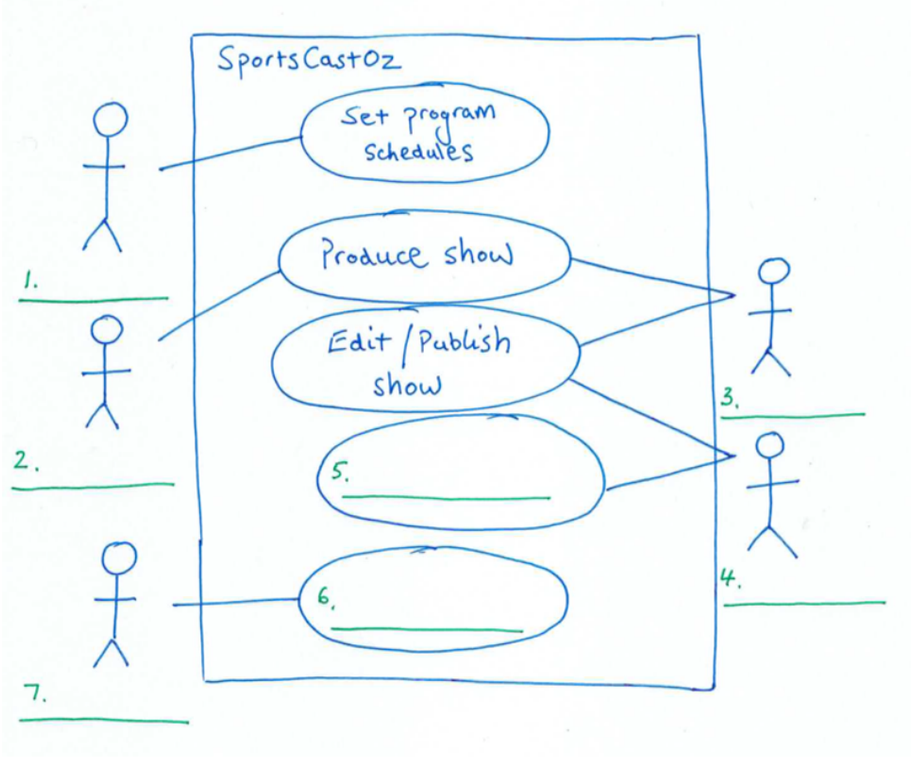


6 marks

Question 9
SportsCastOz is a small Internet radio station that delivers a number of sports based shows either live or via podcast. There are up to 20 presenters involved with the station at any one time. Paul is the director of SportsCastOz and works full-time managing the station and occasionally presenting on the shows. Emily, Traci and Steven work part-time as producers (editing and publishing podcasts as well as managing the live streams). Jay works part time and handles advertising and marketing. Some of this advertising is placed into the edits of the podcasts and some is placed on the website.

Paul would also like to employ someone to develop an App for the radio station. The App would allow the user to listed to all of the shows, read articles published by people at the station as well as provide reviews and feedback.

Paul has started a sketch of a Use Case Diagram to help visually represent what people are doing and how their roles can overlap. Some information is missing. On the diagram below, label the **five** actors and **two** use cases.



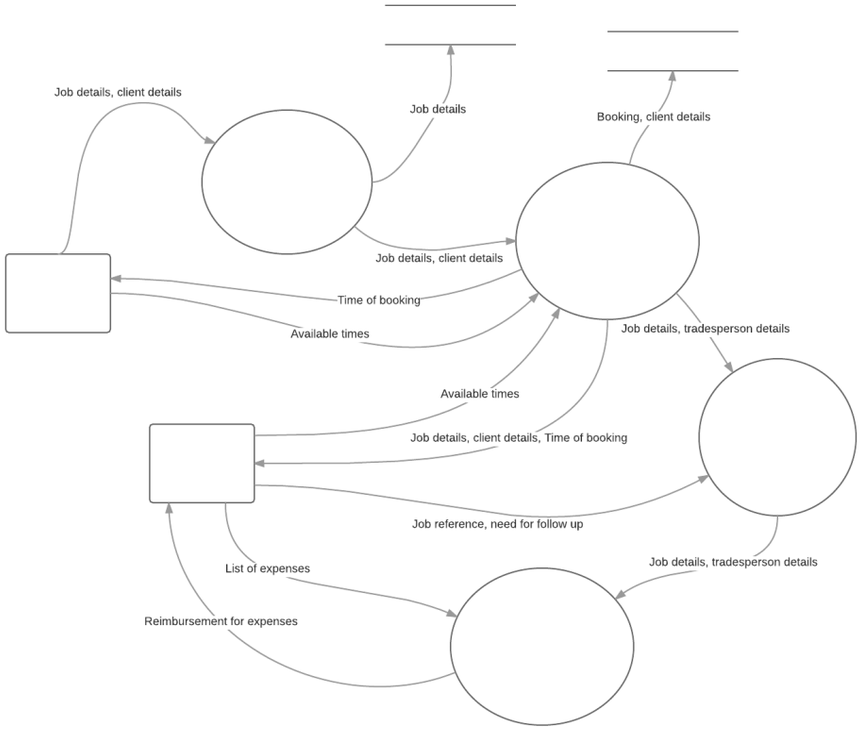
7 marks

Question 10

Nancy wants to create a DFD to represent the maintenance process at her job (*WeCare*). She tries to understand how the job booking system works by drawing a data flow diagram based on the following information.

WeCare have a help line and (for convenience), requires that all maintenance jobs be called in via this line. One of the staff members takes the call and starts a new job (appended to the file of the person making the call). If the person making the call is a new client, then they must be registered with *WeCare* – which requires a detailed application to be completed. At present, the job file is a MS Word document placed into a folder with the client's name on it. A time is booked (that is suitable for the client) for a tradesperson to visit and attend to the problem. The tradesperson then advises *WeCare* of the success (or otherwise) of the job and whether a follow up is needed. Though the tradespeople work on a volunteer basis, *WeCare* does cover the expenses that the trades-people incur, so this is taken care of at the end of the job.

On the diagram on the next page, label the **two** entities, **two** data stores and **four** processes.


Job details, client details
Job details
Booking, client details
Job details, client details
Time of booking
Available times
Job details, tradesperson details
Available times
Job details, client details, Time of booking
Job reference, need for follow up
List of expenses
Job details, tradesperson details
Reimbursement for expenses

7 marks

Sample Examination Answers

Question 1
Answer: D

Question 2
Answer: A

Question 3
Answer: D

Question 4
Answer: D

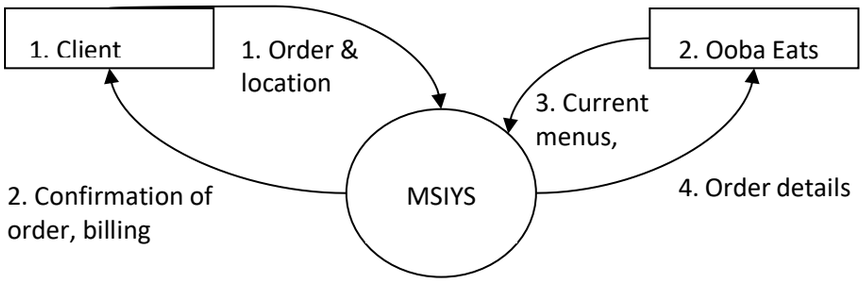
Question 5
Answer: C

Question 6
Answer: A

Question 7

Item Description	Process or Data Store
Combine this year's financial figures with those of previous years.	Data store
Retrieve the number of items from the inventory database.	Data store
Archive the month's data for future reference.	Data store
Discard the daily calculations.	Process
Calculate the profit margin for each stock item.	Process

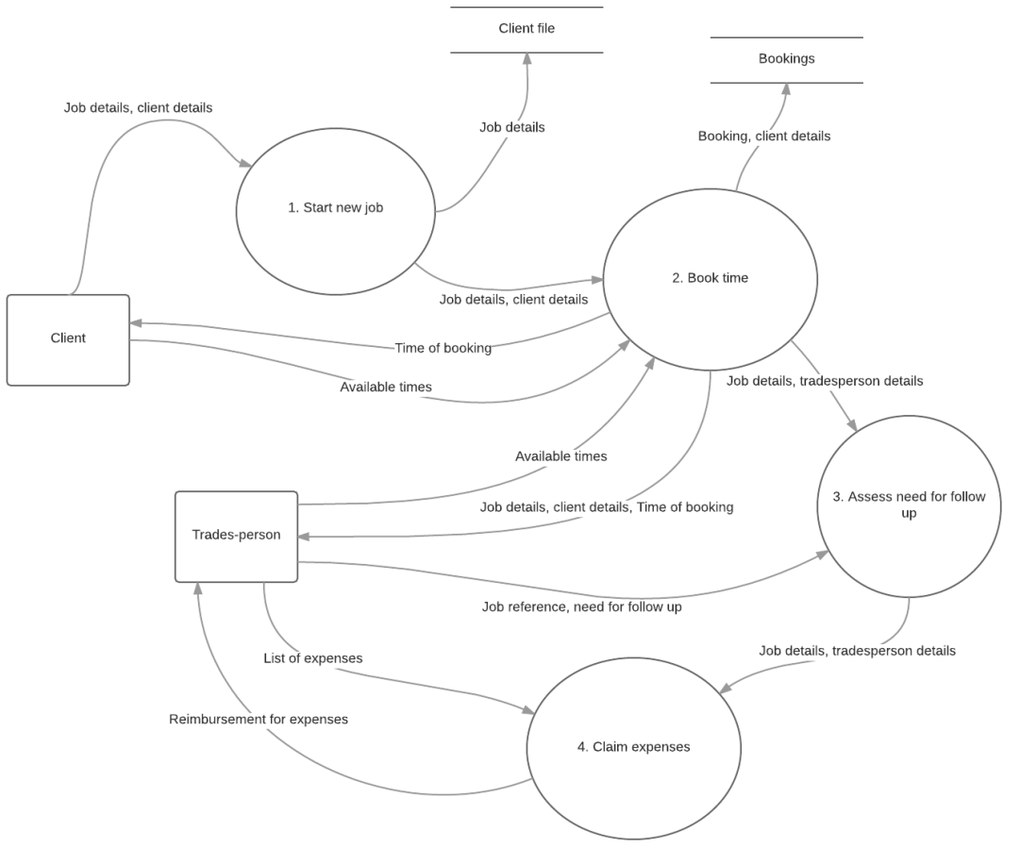
Question 8



Question 9

1. Director (actor)
2. Presenters (actor)
3. Producers (actor)
4. Advertising / Marketing person (actor)
5. Organise advertising (use case)
6. Listen to programs (use case)
7. Listener (actor)

Question 10


Client file
Bookings
Job details, client details
Job details
Booking, client details
1. Start new job
2. Book time
Job details, client details
Client
Time of booking
Available times
Job details, tradesperson details
Available times
3. Assess need for follow up
Job details, client details, Time of booking
Trades-person
Job reference, need for follow up
Job details, tradesperson details
List of expenses
Reimbursement for expenses
4. Claim expenses

Chapter 6
The art of design

The chapter covers Unit 3: Area of Study 2 key knowledge:

- *Digital systems*
    - *KK2.1 Security considerations influencing the design of solutions, including authentication and data protection*
    - *KK2.9 techniques for generating design ideas*
    - *KK2.10 Criteria for evaluating the alternative design ideas and the efficiency and effectiveness of solutions*
    - *KK2.12 Factors influencing the design of solutions, including affordance, interoperability, marketability, security and usability*
    - *KK2.13 Characteristics of user experiences, including efficient and effective user interfaces*

Key terms: Design, computational thinking, brainstorming, DeBono hats, mind-mapping, attribute listing, SCAMPER, efficiency measures, speed, functionality, cost of file manipulation, affordability, data protection, authentication, interoperability, marketability, processing efficiency, effectiveness measures, completeness, readability/clarity, accuracy, timeliness, relevance, attractiveness, accessibility, communication, usability, needs of users, user interface, clear, concise, responsive, familiar, consistent, scalable, forgiving, development time.

Study roadmap

	Unit 3		Unit 4	
Ch	Area of study 1 – Programming	Area of study 2 – Analysis and Design	Area of study 1 - Development and Evaluation	Area of study 2 - Cybersecurity: software security
1	✓			
2	✓			
3	✓			
4		✓		
5		✓		
6		✓		
7				
8				
9				
10				

The art of design

Design is a major stage of the PSM. In fact, the amount of work that needs to be done in the analysis and design stages is significantly greater in terms of tasks to be completed as compared to the development and evaluation stages. Both analysis and design could be shortened simply by performing less actual analysis and spending less time designing the software solution, but doing this is ultimately detrimental to the quality of the solution.

The art of design can be represented by the term ‘computational thinking’. As the glossary definition explains, computational thinking encompasses the ability to think in a logical, creative fashion while having an understanding of the ways in which software solutions and systems perform.

There are a wide range of tools that can be used to assist in the design of a software solution. Not all of the tools described in this text need to be used at once. A software developer will make use of those tools that are the most appropriate for the information

problem as well as those tools that they are most familiar with.

> *Computational thinking: the process of recognising aspects of computation in the world and being able to think logically, algorithmically, recursively and inferentially. It typically involves inferential thinking, defining problems through decomposition, documenting steps and decisions through algorithms, the use of programming languages and software, and evaluating the resulting solutions.*
>
> *VCAA VCE Computing Study Design 2020-2023 Glossary*

Generating design ideas

At the core of design is the process of generating design ideas. It is often a very undervalued step, but producers of successful software development projects will likely espouse the value that the design stage had in the development process.

There are many ways that design ideas can be generated, but we will cover a few key ones.

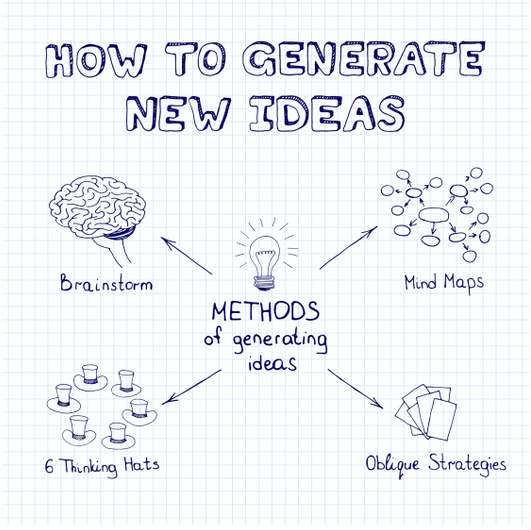


Figure 70: Some methods for generating design ideas

Brainstorming

Brainstorming can be a collective or individual process in which ideas are written down without critical evaluation or criticism. Some commentators believe that group brainstorming sessions are not very effective due to factors such as people within the group blocking others ideas or dominating the discussion. For this reason, it is very important to lay a foundation where each person's contribution is valued. It is also important that suggestions are noted when they occur and are not debated or assessed on the spot. All potential ideas should be noted as they occur, otherwise they can be lost.

For effective brainstorming, the following should ground rules should be established.

- There should be a relaxed environment in which everyone is encouraged to participate.
- No ideas should be judged or critiqued. In fact, quirky ideas should be welcomed and built upon.
- Ideas should only be evaluated at the end of the session.

A nice way to structure and plan a brainstorming session is to use an existing technique such as the DeBono thinking hats.

DeBono 'Six Thinking Hats'

'Six Thinking Hats' is a book written by Edward DeBono that describes a brainstorming tool that can be used very effectively by groups. It assists in the planning and organising of the thinking within a group by ordering the brainstorming process into a number of defined stages (each designated by a different coloured hat). Each of the 'hats' should be (metaphorically) worn by the participants as they work through the process. Though the six hats are defined a certain way, their order is a suggested one and it may be that the person facilitating the brainstorming session wants to focus the group on particular aspects and so will concentrate on those 'hats' alone. It could also be the case that while the session is being

conducted, discussion moves quickly to a conclusion rendering some of the 'hats' redundant.

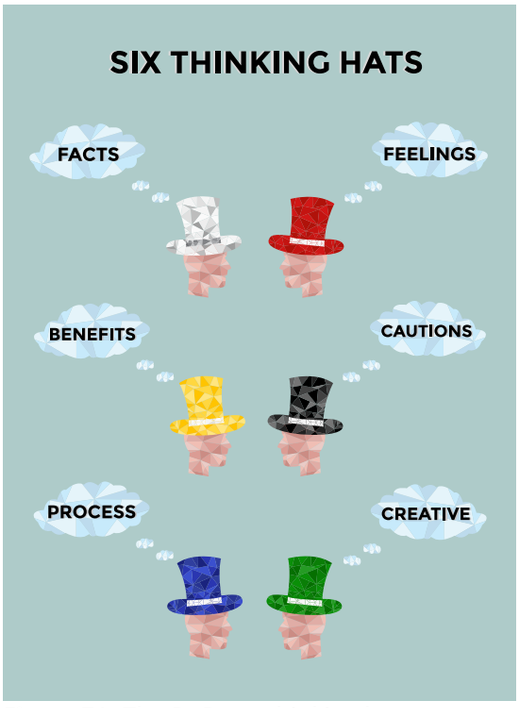


Figure 71: The DeBono thinking hats

White hat (information)

A brainstorming session using the DeBono thinking hats will typically begin with an extended period with just the 'white hat'. The white hat is all about information. What information is available? What are the facts?

Blue hat (managing)

The 'blue hat' is also sometimes used as the first hat as it is concerned with the management of goals and objectives. Using the blue hat, a facilitator can define with the group the way that the brainstorming session will run. Once the brainstorming is underway, the blue hat is used to define and list the goals and objectives of the project or problem.

Red hat (emotions)

While wearing the 'red hat', participants can give instinctive or gut reactions to the problem or the suggestions that have been made. A group facilitator will often try to limit the time spent in this phase as it is often not a very productive one. It can be revisited at different times in the brainstorming process. There does not need to be any justification to responses in this phase and the facilitator needs to ensure that the group understands that this is the expectation.

Black hat (discernment)

While it may seem that the 'black hat' would represent negativity, it in fact indicates a time for the group to consider reasons to be cautious.

Green hat (creativity)

When wearing the 'green hat', group members generate ideas, new concepts and possible solutions. No evaluation or criticism of these ideas takes place during this phase.

Yellow hat (optimistic response)

The ideas that have been presented are now assessed for their relative benefits. Given that the 'yellow hat' is an optimistic response to the ideas that have been presented so far, just as with the green hat, no negative evaluation or criticism should take place during this phase.

One of the key benefits of using a technique such as the DeBono thinking hats, is that it helps define the rules and structure to the brainstorming session. Often brainstorming sessions will break down or be less effective as some participants will be offering suggestions (green or yellow) while others are discussing goals and information (blue or white) or critically evaluating the ideas being presented (black).

Oblique Strategies

The set of cards known as 'oblique strategies' is intended to promote lateral thinking and break groups out of thinking in set ways. Developed in 1975, there are 55 cards in the

set and they have been used successfully in very creative fields such as music and art.

Some examples are:

- 'What to increase? What to reduce?'
- 'State the problem in words as clearly as possible'
- 'Use an old idea'

Mind mapping and graphic organisers

Mapping tools and graphics organisers allow those involved in the brainstorming process to visually represent ideas and the relationships between them. Drawing a mind map is an excellent method of visually representing what is known about a topic or a problem. The process of creating a mind map helps those creating it to understand what they know about a topic, what questions still exist and how to group different facets together.
Many software tools exist though it is often the case that people will prefer to use pen and paper (or whiteboards) as it is easier to represent ideas quickly.

Figure 72: A person working on a mind map

A mind map is a type of diagram that has at its centre the main idea or concept. Related concepts are then connected to the central one by a line and the diagram grows – branching out in a circular pattern.

Mind maps can be especially useful as a reference tool. They can also be used to communicate the scope and constraints of a proposed solution to others.

Researching

Researching ideas of concepts that are already in existence can also be a very valid method of generating design ideas. It can also form the basis for other brainstorming methods to build on the inspiration that these ideas may provide.

Attribute listing

The technique known as 'attribute listing' is also one that works best in concert with other brainstorming techniques. In essence, it involves listing the different possible attributes related to aspects of a problem and then mixing and matching these in ways that may create an innovative solution.

For example, a company may wish to design a new software solution that customers will access to order goods. The first step in utilising this technique, is to list the attributes that such a software solution might have.

For example:

- Storage type
- Platform
- Programming language
- Authentication method.

For each of these attributes, the options that are available are listed below each other inside a table.

The attribute table can then be used to 'mix and match' combinations of the attributes to see if one of the combinations is a viable design alternative to consider.

SCAMPER

SCAMPER is an acronym for a structured brainstorming technique that encourages users to think outside of the box. The user first selects a design from which the alternative

designs will be generated. The letters of SCAMPER guide the process from this point onwards.

It is important to be aware that this technique is primarily used for the improvement of an existing design and needs a starting point from which to proceed. It could be that the starting point for the SCAMPER has been developed using one of the other methods for generating design ideas. The letters of SCAMPER stand for the following:

S stands for Substitute
Substitute components of the design, materials, people/characters, approach, angle, colour, shape, meaning, idea or words.
Questions that can be used to stimulate discussion: What can you substitute? What/who can be used instead? What else can be used instead? Are there other materials? Are there other processes? Are there other shapes? Can another place be used? Is there another approach?

C stands for Combine
Mix or blend the design with other things or ideas.
Questions that can be used to stimulate discussion: What can you combine or bring together? How about a blend, a mixture, an assortment or an ensemble? Can you combine the purposes of two or more designs?

A is for Adapt
In what ways can you alter, change the function or use a part of another element?
Questions that can be used to stimulate discussion: What can you adapt for use as a solution? What else is like this? What other ideas does this suggest? Do past designs offer a parallel? What existing designs could I copy? Who could I emulate?

M is for Modify (Magnify or Minify)
Increase or decrease the design in scale, change shape, change attributes such as colour or give it a new angle.
Questions that can be used to stimulate discussion: Can it be duplicated, multiplied, exaggerated? Can anything be omitted, made shorter, smaller, lightened? Can you change the meaning, colour, motion, sound, smell, form, shape? Consider magnifying aspects of the design. What can you add or exaggerate? Consider minifying aspects of the design. What could you make smaller or condensed?

P is for Put
Put the design or aspects to another use or idea.
Questions that can be used to stimulate discussion: How can you put the design to different or other uses? Are there new ways to use the design as is? Are there other uses if it is modified?

E is for Elliminate
Eliminate or remove elements in order to simplify the design.
Questions that can be used to stimulate discussion: What can you eliminate? Rather than thinking of specific elements of the design, what other things could be removed? What about if time, effort or costs were reduced?

R is for Reverse (or Rearrange)
The design could be turned inside out or upside down, inversed, components could be swapped or transposed.
Questions that can be used to stimulate discussion: What can be rearranged in some way? Are there components in the design that can be interchanged?

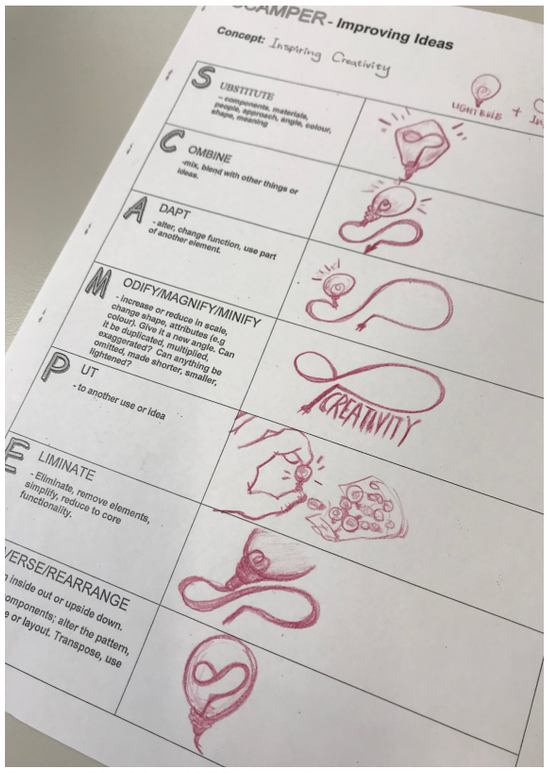


Figure 73: SCAMPER in action

Efficiency and effectiveness in the design phase

Having an understanding of the different efficiency and effectiveness measures that are used to assess a software solution is important to the design phase. Efficiency and effectiveness can be used when designing the user interface of the software as well as the criteria by which designs will be contrasted and compared. Let's examine some of these measures in detail.

Efficiency measures for a software solution

Speed of processing

Is data able to be entered quickly? Is the output from the software solution generated in a prompt fashion? The layout of the user interface and the way that the software flows from one screen to another can also have a profound effect on speed.

Functionality

How well are the functions of the user interface designed and how well do they address the tasks for which they have been designed?

Cost of file manipulation

Each time a file needs to be manipulated in some way, the efficiency of the overall solution will be affected. Processing the data within a file, sorting, searching or moving the data from one file or location to another will all have a time cost associated with it. The challenge is to minimise this cost in any way possible.

Effectiveness measures for a software solution

Completeness

How complete is the information that is being produced for the purposes that it is intended? Does the user of the software need to source additional information from other software solutions or sources? Completeness is a subjective measure based on the task that is being performed and the user's needs – so a detailed analysis of this should have been performed to prevent a lack of completeness occurring.

Readability or clarity

How easy is it to read and understand the output that is produced? If the information is not easy to read, the effectiveness of the software solution is lessened even if all of the requested information has actually been produced.

Attractiveness

How attractive is the user interface? This can have a flow on effect in terms of the usability of the software solution. Attention to the colour scheme, font size, consistency of object sizes and access to the program's functions can all have a positive effect on the user's

experience and as a consequence, make the program more effective.

Accuracy

Is the information produced always correct? Does it often need to be adjusted or queried? Have difficulties occurred in the past due to the inaccuracy of information? A well designed user interface can ensure the accuracy of the inputted data by limiting the type and range of the data. Validation techniques can detect when errors occur and assist users in making corrections.

Accessibility

To what extent are the functions of the user interface easy to find and use? If users of a software solution are confused by the function of particular elements, it will slow them down and make the software not as effective. Functions, icons and navigation should clearly indicate their function without confusion. Help files and on-screen help assist with accessibility.

Software developers need to be careful to design their solution with the users of that solution in mind. The skill level of the users as well as the hardware and environment that the solution will be used on, can all be factors in limiting the accessibility. If a software solution is designed for the latest computer with a large amount of RAM and a high screen resolution, but will be used by an organisation in which the typical hardware is three years old and the screen resolution is smaller, it will make the software very difficult to use.

The measure of accessibility is also concerned with ensuring that the software solution caters for those with disabilities such as vision impairment, by adding features that account for these disabilities – such as allowing text to be resized.

Timeliness

Is the information produced within a time frame that is relevant and useful? That is, by the time the information has been produced is it ever too late to make practical use of it? Data such as stock prices and weather conditions are affected by the attribute of timeliness and are often used by software solutions to make predictions or strategic decisions in a very time dependant way. This data not only needs to be able to be entered into the software quickly and easily, but the processing of the software needs to produce the required information in time for it to be used.

Communication of message

How clearly is the software solution presenting the information that it is producing? Is this information hard to interpret or is it easy to understand? The information may be accurate, timely and relevant, but simply presented in a way that makes it difficult to understand or not well suited to the role of the user within the organisation. For example, a manager may wish to extract information from a software solution that shows them how profitable the company has been in the last month. The software solution may have all of the required information, but the manager may simply want to be presented with a graph or bottom line figure rather than reading through all of the companies transactions for the month.

Relevance

Does the information contain the necessary detail or elements? When a user tries to extract information from the software, is the information that is presented to them relevant to their request? Relevance can also be improved by providing user interface features that target the user's level of access and role. Validation techniques that filter out incorrect or inaccurate data can help to improve the relevance of results that are produced.

It is certainly true that information is very valuable and organisations will never discard information (preferring instead to archive it away from the central storage location). As more and more information is produced, there could be a tendency for a software solution to present so much information to the user, that

the information that they were seeking is 'lost'. Archiving can certainly prevent this sort of occurrence, but software can also be designed to filter irrelevant information out so the user does not have too much presented to them at once.

When discussing the relevance of the information produced by a software solution, it can also be useful to consider redundancy. For example, a software solution might contain a function that displays all invoices that have not been paid. If this information is presented as a list on the screen with the heading 'Invoices not paid' and next to every item is the text 'not paid', then the user is being presented with a lot of information that they already know.

Usability

Is the user interface intuitive and easy to use? Well designed software should have a user interface that acts as a conduit to the required processing but does not become so much of a focus that it detracts from the functioning of the software itself. Sometimes the use of too many functions on the screen at once or the use of animations can detract from the usability of the solution.

There is a definite art to the creation of a user interface, but the software should be 'about' the task itself and not the interface.

Factors influencing the design of solutions

Usability

Following on from the point above which describes usability as an effectiveness measure, it is also one of the key factors that needs to be taken into account in the design stage. When designing a software solution, the users themselves are of prime importance. If a solution does not cater for the specific needs of the users, then the solution cannot be considered a success, even if it accomplishes the tasks demanded of it.

The amount of experience that the users of the proposed solution have can also have an effect on the types of functions that are included in the design. For example, if the users of a proposed software solution are highly capable IT users with a lot of experience, it may be appropriate to build extra functions into the solution that allow them to accomplish quite complicated tasks. They may be given the ability to write scripts or create macros within the solution. If the users did not have the knowledge or expertise to do these sorts of things, it would not be a good use of time to build this into a solution.

Affordability

Often the scope and features of a proposed project will ultimately be limited by the amount of money that is available in the budget. With unlimited funds and time, a solution to almost any problem could be conceived, but this is not realistic. A design that does not take the available budget into account can doom a project to failure.

Security: data protection and authentication

It goes without saying that any software solution will contain security features to protect the data it stores and ensure that those accessing it have the necessary levels of authority. Legislation ensures that all organisations enact good data protection practices. The most relevant legislation is the Privacy Act, but there are many that have

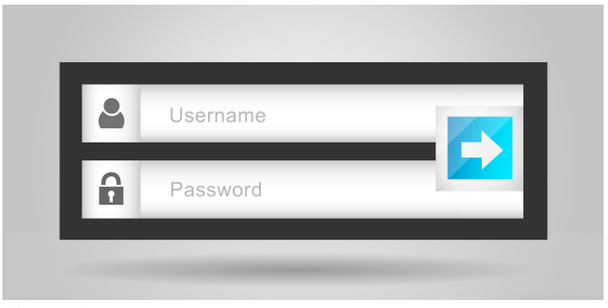


Figure 74: Login screen

bearing on the ways in which organisations use, store and display their data.

While this is a given, a software developer needs to ensure that these aspects are not an after-thought from a design perspective. Designing a user interface that allows the user to sign in efficiently (or perhaps integrates with a SSO – Single Sign On) is important. It is also important to consider what the interface will look like and how the solution will operate with different levels of access enabled.

Interoperability

Designing a software solution so that is works with the other software packages being used on a system is also a key consideration. Software packages will often share data and so need to used file formats that are standards based such as XML.

An API (Application Programming Interface) in simple terms is like a library of routines that can be plugged into a software solution to allow it to share data with another one. In designing a new software solution, the ability to be able to use particular APIs may be critical to allow it to be properly integrated with the existing system or indeed systems in any organisation in which the software may be used.

Marketability

Marketability is the measure of how much something can be bought or sold. In what ways could the marketability of a software solution have an effect on the design? It all depends on what context the software solution is being designed for. If, for example, a software solution is being designed for a small company and is only being planned for use by their employees and on their hardware, then perhaps the marketability of the final product is not important. At the other end of the spectrum would be a software solution that is being created specifically to be sold. Between these two scenarios is probably the situation in which what is being created has the possibility of being marketed at some future point in time.

Processing efficiency

In what ways can processing efficiency influence the design of a software solution? A solution may require information to be available within a certain period of time and the data that needs to be gathered in order to produce this information may be varied in type and form. It may also be the case that the amount of processing that needs to be done is quite large and will have an effect on the time the solution needs to run. Many software solutions that have a variety of data sources and are producing information that is time dependant will present users with different options at different times and prompt users based on what they require.

For example, Parent Teacher Online by Country Net Solutions (shown in Figure 63) presents each user with a menu that is structured so that data is entered in the correct order. Particular options become available only at certain times, preventing users from trying to access information before it has been correctly processed.

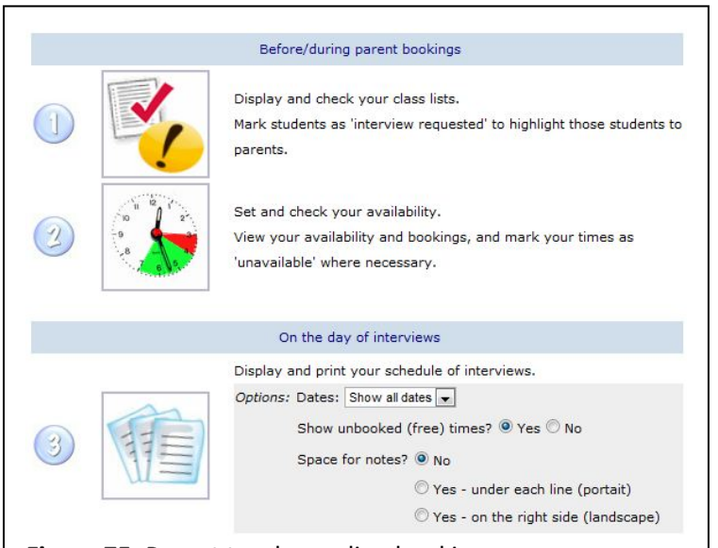


Figure 75: Parent teacher online booking
PTOnline by Country Net Solutions, image courtesy of Country Net Solutions

User interfaces

There is a lot of information available on the creation of effective user interfaces. The creation of a user interface is often such a major part of the development of a software solution, that there will be software developers dedicated to working on this aspect of the solution. When you reflect on successful technological devices of the last few years, they generally will all have one feature in common: a highly successful user interface. A good user interface allows users to access the software solution with ease and perform the tasks required of it. It has been said that if a user begins to 'notice' a user interface, then perhaps the user interface is not doing its job.

Let's look at some of the characteristics of effective user interfaces.

Clear

A clear user interface is one that is easy to read and understand. The interface should communicate its function in a way that does not leave the user wondering what a particular function does. In instances where the user can hover over a button or function or access a help dialog, the explanation should be clear and easy to understand.

Concise

As well as being clear (and possibility in competition with it) is the desire to be concise. Users do not want to be presented with long explanations of the functions within the software solution. Functions and explanations should be as concise as possible, explaining the tasks performed without being so lengthy that they dominate the user interface.

Responsive

When discussing a responsive user interface, we are not just talking about speed. A fast user interface is definitely desirable but this is often influenced by the processing that is being done. A responsive user interface is not only as fast as it can be, but it gives feedback to the user in the form of progress bars, messages, icons that indicate that processing is taking place (hour glass, spinning wheel) and confirmation that tasks have been undertaken or completed.

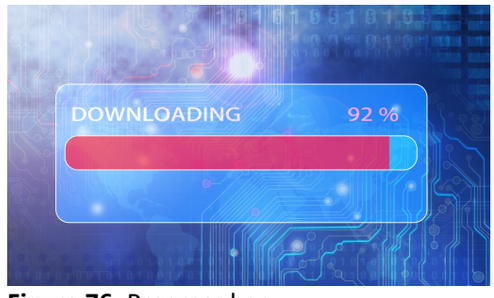


Figure 76: Progress bar

Familiar

A user interface that is familiar in style and operation to those of other successful user interfaces will make users comfortable and mean that they will be able to use the software much more intuitively. The term 'familiar' is a synonym for intuitive in this context, but it can difficult to describe how to make a user interface intuitive. Familiarity does not need to come from other successful user interfaces; it can also come from a comparison with real life objects and the ways that we interact with them. A user interface that is styled as a book, for example, could have controls that turn the pages or add bookmarks that can be referred to later on.

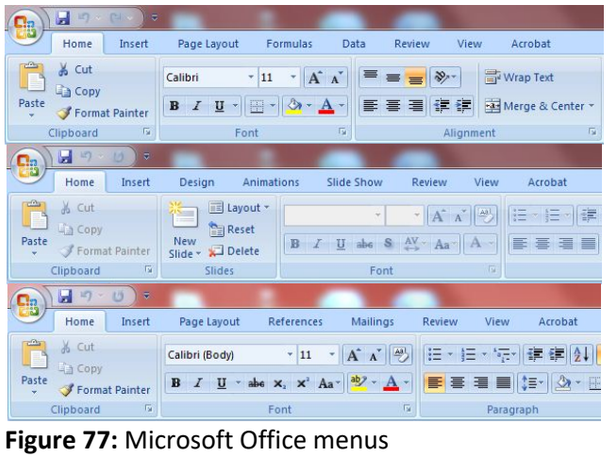


Figure 77: Microsoft Office menus

Consistent

User interfaces should strive for consistency across all of their screens so that the user can easily use what they have learnt in one part of a software solution to the whole product. For example, consistent navigational controls that are placed in the same position each time and operate the same way.

The different software programs in the Microsoft Office package all share a very similar interface that allows users to move from one to the next without having to relearn the menu system in its entirety.

Efficient

An efficient user interface is one that allows the user to execute tasks with a minimum of effort. The task itself may appear easy to complete, but there are some challenges in doing this and simply placing all of the functions that are available to the user on the screen at once does not make the user interface efficient. Microsoft Windows, for example, will present the user with a list of options when they try to save an image from a website, as shown in Figure 67. These options try to anticipate the possible tasks that the user may want to accomplish.

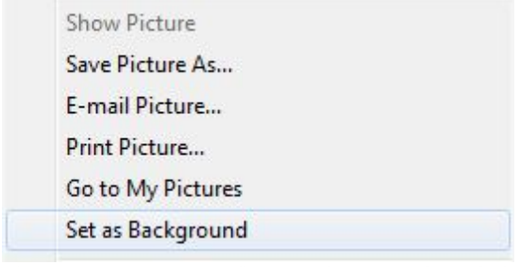


Figure 78: Microsoft Windows right click menu

Attractive

It is easy to underestimate the effect that an attractive user interface has on the user's overall experience. Certainly the combination of colours, fonts, screen size and layout all play a part in the attractiveness of a user interface, but it is more than this. An attractive user interface is enjoyable to use and is tailored to the users themselves. The users at an accounting firm may find a very logical and ordered user interface to be attractive, whereas the users at a fashion house might like a user interface that features lots of bright colours and revolving fashion designs in the background of the screen.

Scalable

A user interface that has been designed to be scalable is one that has allowed for the future addition of functions or features in such a way, as their addition will not adversely affect the overall layout of the software. Although this is probably not high on the list of desirable features of an effective user interface, it is nonetheless one that warrants consideration.

Forgiving

User interfaces that allow the user to roll back any transactions that have been performed are highly desirable. The 'undo' and 'redo' functions are almost considered to be core functions of a software solution by many and the inclusion of them means that users gain a sense of security and don't have to be tentative or hesitate when using the software solution.

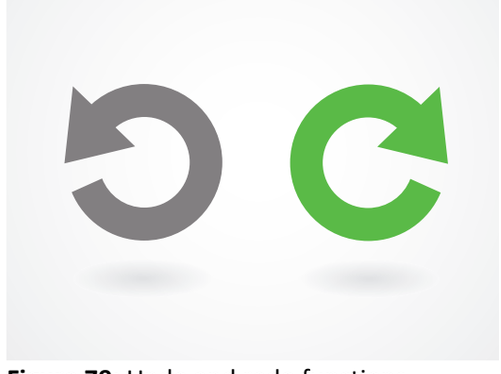


Figure 79: Undo and redo functions

Context Questions

1. A user interface that is described as 'responsive' is more than simply 'fast'. What other attributes does an interface have that is 'responsive'?
2. What does it mean if a user interface is described as efficient?
3. What are the benefits in determining the evaluation criteria for a software solution during the design stage?
4. Considering brainstorming techniques such as DeBono's hats, what are the key ground rules that should be established before participants begin giving suggestions?
5. List and describe 5 measures of a user interface in relation to its effectiveness.
6. In what ways do security requirements influence design?
7. What is the difference between user interfaces being described as 'familiar' as opposed to 'consistent'?
8. The term 'computational thinking' refers to being able to think recursively. What does this mean in a design context?

Applying the Concepts

- Reflecting on your favourite Apps, what aspects of their user interface design are most appealing to you?
- Pick a household object or appliance and perform a SCAMPER design process on it. Each person in the class could do a SCAMPER for a different object and compare results at the end.

Key Skills Checklist

At the conclusion of this chapter, you should be able to address the following key skills. Mark each off as you can achieve them.

Generate new design ideas using one of several methods	☐
List and describe efficiency measures of a software solution	☐
List and describe effectiveness measures of a software solution	☐
Describe factors that influence the design of solutions	☐
Discuss what makes efficient and effective user interfaces	☐

Sample Examination Questions

The following sample examination questions can be attempted to test your knowledge of the content of this chapter.

Question 1
At the beginning of a project, Danny decides that it would be good for the software development team to brainstorm possible solutions. Which of the following would be a valid brainstorming technique?

- **A.** Creating a mock-up
- **B.** Writing a pseudo-code algorithm
- **C.** Drawing a mind-map
- **D.** Researching current trends and products

Question 2
In creating a mind map, which of the following is the most correct?

- **A.** The central idea or concept is placed in the middle of the mind map
- **B.** Each person thinks of their own ideas and these are combined at the end
- **C.** More important concepts are represented with larger circles
- **D.** A maximum of 6 circles should be added to ensure that the ideas are not too broad

Question 3
A user interface that is described as being 'responsive' is:

- **A.** One that is fast above all else
- **B.** One that gives the user good instructions
- **C.** One that is not only fast, but gives good feedback to the user
- **D.** One that is able to print to a number of different network devices

Question 4
An interface that is 'forgiving' is one that:

- **A.** Works reliably all of the time
- **B.** Is easy to understand
- **C.** Allows the user to undo transactions
- **D.** Works on a variety of platforms

Question 5
Phillip is to use a method for generating new design ideas for an App he is developing. Which of the following is not a method he could use?

- **A.** Brainstorm
- **B.** Use Case Diagram
- **C.** DeBono's thinking hats
- **D.** Oblique strategies

Question 6
There are a number of ways to generate different design ideas. Describe **two** methods that can be used and explain which of the two you would recommend for use by a local council attempting to solve the issue of crime in the area?

Method 1: _____

Method 2: _____

Recommended: _____

_____

_____

3 marks

Question 7

Jill's Green Thumbs is a garden maintenance business developing an App that will allow customers to submit a request for a quote. It will:

- allow them to enter their contact information
- use location services to pinpoint their exact map location
- take photos, annotate them and add them to the detail of the quote
- connect to a database of available plants and materials and their cost
- allow the user to create sketches as needed.
- email the finished quote request to Jill's Green Thumbs

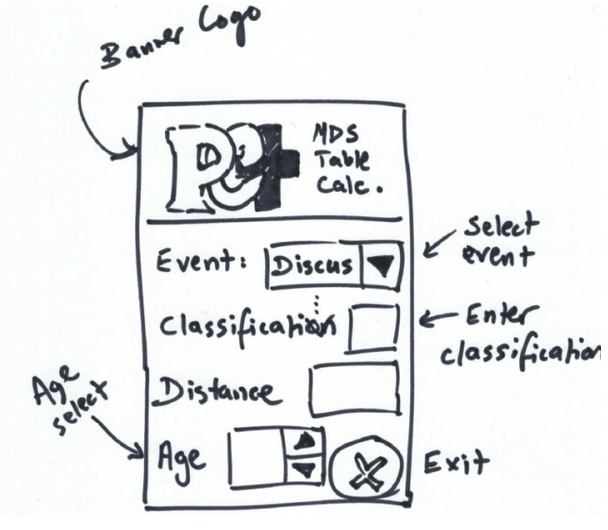
In the table below, suggest a feature of the user interface for the App in each category.

Clear	
Responsive	
Familiar	
Efficient	
Forgiving	

5 marks

Question 8

The annotated screen design below is for a proposed App that will be used in junior athletics to convert the results of throws from para-athletes using MDS tables.



a. Despite the efficient design, when the developers seek feedback from their peers, many tell them that it is not 'scalable'. What does this mean and how could the design be changed to make it 'scalable'?

_____

_____

2 marks

b. Name two elements of the design shown that are 'familiar'?

_____

_____

2 marks

Sample Examination Answers

Question 1
Answer: C

A and B require some decision to have been made on the direction / purpose of the App. D is not a valid choice either as it is researching the ideas of others rather than coming up with one of your own.

Question 2
Answer: A

Question 3
Answer: C

Question 4
Answer: C

Question 5
Answer: B

Each of the other three options are tools for generating design ideas. A Use Case Diagram is a method of representing the processes within an existing system.

Question: 6
Method 1: Brainstorming – ideas are written down as they are offered and then the group goes through them critically.
Method 2: DeBono's six thinking hats – a group brainstorming session is structured by focusing everyone on facts (white hat) firstly and then moving through a number of phases.
Recommend: A complex issue such as crime in a local area might be well served by having a discussion using DeBono's thinking hats. This way the discussion can remain focused at all times.

There are a number of possible answers here. The two methods could include mind mapping or some other method of generating design ideas. The method that is recommended is also not important – the focus is on explaining why the method chosen would be a good choice.

Question 7

Clear: All aspects of the user interface should be easy to find and understand. It should be easy to see where to start with the quote and how to submit it.
Responsive: Upload photos and information quickly and give the user an indication of what processing is taking place.
Familiar: Use standard icons for features such as location services and email.
Efficient: Allow the user to complete their quote in as few steps as possible.
Forgiving: Allow the user to undo or go back to modify steps in their quote.

Other options are possible. When answering questions such as this in the exam, it is important to put your answer into the correct context. Make sure that you mention aspects from the question or case study in your answer.

Question 8

a. There isn't enough space on the screen to expand the functionality if required. It could be fixed by having a greater amount of white space or introducing a menu or tabbed interface.

b. Name **two** elements of the design above that are 'familiar'?
The exit button (X) and the drop down lists.

Chapter 7
Of input and output

The chapter covers Unit 4: Area of Study 1 key knowledge:

- *Digital systems*
  - *KK3.1 Procedures and techniques for handling and managing files and data, including archiving, backing up, disposing of files and data and security*
- *Data and information*
  - *KK3.2 Ways in which storage medium, transmission technologies and organisation of files affect access to data*
  - *KK3.3 Uses of data structures to organise and manipulate data*

Key terms: Archiving, backup, full backup, differential backup, incremental backup, backup media, RAID, disposal, cloud storage, tape backup, hard disk, optical storage, solid state drive (SSD), data structure, array, associative arrays, classes, fields, files, hash tables, linked lists, queues, records, stacks.

Study roadmap

	Unit 3		Unit 4	
Ch	Area of study 1 – Programming	Area of study 2 – Analysis and Design	Area of study 1 – Development and Evaluation	Area of study 2 - Cybersecurity: software security
1	✓			
2	✓			
3	✓			
4		✓		
5		✓		
6		✓		
7			✓	
8				
9				
10				

Handling files and data

A number of procedures and techniques are often used when managing files and data. As files are easily corrupted, it is often best to leave them 'open' for the shortest time possible. From a programming perspective the best way to do this is to 'open' the file, read in its contents and then 'close' the file. If the file is 'open' and the software solution experiences an unexpected crash, it is quite likely that the file will be damaged. Opening files for prolonged periods can also mean that other users cannot access the file.

Input files should be located in a folder that is specific to the software solution so that they are easy to locate and are grouped with the solution when files are backed up. This can also be achieved by including an import feature in the software that copies a file from any location to a secure folder within the solution.

Naming of files is also important. A specific file extension may be chosen so that files are not misinterpreted as belonging to another software package. Files should be named in ways that identify them and distinguish them from previous versions.

Of specific importance when discussing the handling and management of files and data are issues of security, archiving, backing up and disposal.

Security of files

A common use of files is the storing of sensitive information such as usernames and passwords or logs of user activity. It can be a dangerous practice to leave these files unprotected, as someone could choose to access the file directly rather than through the software solution, potentially bypassing any security features of the program. The easiest way to protect files of this sort is to encrypt them – a process that is described in more detail in Chapter 10. Files can also be password protected or placed into folders that are only accessible by those with administrator permissions.

Archiving

Archiving is a task performed by all organisations at some time. As the amount of storage space that is available for software and data is finite, archiving is necessary to remove those files that are no longer needed for immediate access. The size and number of files that are being used on the main storage media also has a significant effect on the backup routine – as a large amount of data is much harder to backup than a small amount (see below). Archiving is simply the process of removing those files that are no longer required to be accessed to another storage medium that can securely store them for an indefinite period of time. Should the files in an archive need to be accessed, they should be able to be opened via a process that makes them available to the software.

Backing up

Data and information within an organisation are exposed to a number of threats. These threats can be as simple as the accidental damage or loss of storage devices, to deliberate acts that destroy, delete or corrupt data. The process of backing up the data and information within an organisation is vital to ensure that the operation of the organisation continues as unaffected as possible in the case of a data loss of this kind.

Version control is paramount to managing the backups that are being produced in an organisation. Version control software (VCS) packages are applications that assist organisations to keep track of the versions of their backups / files.

Performing a backup can be as simple as selecting a number of files and making a copy of them that can be placed in a secure location. Organisations are generally much more organised than this and will have a specific backup routine that they follow regularly. The extent and frequency of this routine is often determined by how much data there is to backup, what facilities are available and how valuable the data is.

The use of an Uninterruptable Power Supply (UPS) can keep a server running in the event of a power outage for enough time for an up to date backup to be made and for the system to be shut down in a safe manner that doesn't damage any files.

There are three different types of backup: full, differential and incremental.

When safes are labelled 'fireproof', this often means that they are 'flameproof' and will protect their contents for a certain period of time. Higher quality safes are also 'waterproof' so that the contents are not damaged by the water that is used to put out a fire that is threatening the safe.

Full backup

A full backup is the simplest method of backing up data. When a full backup is performed, all of the data is copied to the backup media. This can be time consuming if there is a large amount of data and requires the use of a backup medium that can accommodate the combined size of all of the files (which can be expensive). The advantage in using a full

backup comes when the files need to be restored. As a full backup is a complete set of all of the organisation's data, it can be simply restored all at once or aspects of it selected.

Differential backup

A differential backup is used in conjunction with a full backup. A full backup is made and then subsequent backups contain only those files that have been changed (since the full backup). Note that there are technically only two backups when using this method (although more copies may be made). The full backup is the first and the second is the most recent differential backup. This makes the process of completing daily backups much simpler and quicker, but does mean that the process of restoring the backup is more complex than restoring a full backup. To fully restore the backup, the full backup would need to be restored, followed by the most recent differential backup.

The size of the differential backup will grow as the number of files that have been modified from the last full backup also grows. A point will be reached when it becomes practical to make another full backup to reduce the size of the differential backup. To get around this problem, an incremental backup can be used.

Incremental backup

An incremental backup works in a similar way to a differential backup. A full backup is made to start the process. Subsequent backups then consist of any changes that have occurred since the last incremental backup. This makes the backup process very fast and efficient. An incremental backup is more complicated to restore however, as the full backup needs to be restored followed by all of the incremental backups in order. For this reason, a logical file naming convention is needed to ensure that the order of backups is maintained.

When to perform a backup?

Backups are generally performed as automated processes at the end of the working day. It is important that the users of the system have finished their work and closed all of their files, as this can interrupt or impede the backup procedure. A backup procedure can also take a long time to complete. For these reasons, backups are often set to be performed during the night.

As they are often automated processes, the backup media have to be able to accommodate the size of the backup without needing to be swapped or changed.

How often to backup can be a personal choice. There is a saying that you should only backup that which you cannot afford to lose. At the same time, there has to be a balance between the time spent backing up and the time spent working. Daily backups are commonly implemented by companies, but this could be varied based on the size of the company and the amount of data being processed.

Backup media

There are a wide variety of hardware devices that can be used to store backup data. The choice of device will depend on the capacity required, access speed, convenience and cost. Capacity is of particular importance as the backup media should ideally be able to backup all of the data without the media needing to be swapped. Swapping media while in the process of backing up will complicate the process and make it less likely that the backup will be performed correctly each time. It is also harder to automate a backup process that involves swapping the backup media.

Cloud storage

When it comes to backup media, many organisations will make use of cloud storage solutions. This may be via a formal backup procedure that is moving data to the cloud storage site on a regular schedule or simply manually moving data to one of the many

cloud platforms such as Google Drive or Microsoft's OneDrive.

The advantage of using cloud storage as a backup media is that it is independent of physical media and relies only on the accessibility of the cloud server. There is little infrastructure that the organisation needs to implement. Retrieving a backup is as simple as connecting to the cloud server and transferring the required files. The only disadvantage of using this form of backup media, is that an organisation will be unable to access the data in the event of a power outage.

Tape backup

Tape backup is still used today despite it being one of the oldest forms of backup media available. Some organisations utilising tape backup systems will be doing so as it is a legacy system and all of their previous backups are in this form. Its use is certainly declining, but it will probably be used for a number of years still based on the advantages that it provides.

Tape is very good at holding onto data for really long periods of time and is still relatively cheap. Many tape formats offer encryption features, which allow for easier compliance with various laws and information security standards. In addition to this, a tape backup means that an organisation maintains physical control of the data, which may cause organisations to opt for media like this over cloud storage.

Tapes are portable, small and manageable. They are easy to store compared to hard drives, servers and other physical media. They are also very durable. Many digital tapes can retain data for 30 years or more without regular maintenance. Given this and the fact that the cost is relatively low, an organisation can archive data without the need to reuse backup media.

The capacities of tape drives are increasing each year. The current standard (LTO-8) can store 12 TB and 30 TB if the data is compressed. Data can be accessed at 320 MB per second data rate.

Hard disk

The price per capacity for hard disks has been making them more competitive as a backup medium in recent years. The main advantage that hard disk storage has over tape storage, is access time. Performing a backup to a hard disk is relatively quick and retrieving data from the disk is equally so. Capacities of hard disks are now in a similar range to that of tapes. In addition to this, hard disks are highly portable, can be connected to an information system in a wide variety of ways and they are a physical form of media that once again, allows an organisation to maintain control of its data.

Figure 80: Backup tape LTO storage installed in a data centre server room

The main disadvantage of hard drive storage is its volatility. Being a mechanical device, it is more easily damaged (especially while being transported). Some hard drive manufacturers have included technology to support a drive's contents in the event of it being dropped and others encase their drive in a shock absorbing case or surround.

Despite these improvements, drives can be easily damaged and manufacturers do not make any guarantees about the safety of the data stored on their drives in the event of a drop or physical event.

Hard drives also do not retain their data for long periods of time. Given their cost, they can be replaced with newer drives, but organisations would need to be purchasing new hard drives in the region of 3-5 years.

Figure 81: Inside a hard disk drive

Optical storage

Optical storage has been on a rapid decline with the increased use of cloud storage and the cheap cost of portable hard drives. Recordable CDs, DVDs, and Blu-ray discs were common forms of backup, especially for consumers, due to their low cost. Each of the optical storage media listed above had limited capacity and were also time consuming to write data to. Interestingly, tests conducted on standard consumer grade optical media has shown that it has an archival period of up to 10 years. High end optical media produced with a special gold-sputtered layer is projected to have a retention as high as 100 years.

Solid-state drive (SSD)

Solid-state drives (also known as USB flash memory, thumb drives etc.), are devices that are relatively expensive for their low capacity in comparison to hard disk drives, but are highly portable and easy to transport. Unlike a hard disk, a solid-state drive does not contain any movable parts making it less susceptible to physical damage. Their speed of access is determined by the interface, but is generally in the range of 500Mbit/s to 6Gbit/s. Capacities continue to grow.

While SSDs can be very convenient to use, their use as a backup media is questionable. Given the short amount of time that SSDs have been in mainstream use, information concerning their ability to retain data is limited. Their cost per unit of storage makes them one of the most expensive forms of media storage, and when an SSD fails, data recovery is difficult.

Backup Media

Backup Media	Type	Capacity	Cost	Max write speed
Optical media	CDR	~700 MB	~$0.30 per disc	~7 MB/sec (52x)
Optical media	DVD	4GB	~$0.60 per disc	~21 MB/sec (16x)
Optical media	Blu-ray	25 GB / 50 GB (single / dual layer)	~$3-$8 per disc	~72 MB/sec (16x)
Hard disk	Internal	1-14 TB	$50 - $1000	~200 MB/sec
Hard disk	USB	1-5 TB	$50 - $400	~130 MB/sec
Solid state drive	Internal disk	120 GB – 4 TB	$30 - $1000	~550 MB/sec
Solid state drive	USB	16 GB – 512 GB	$10 - $400	~300 MB/sec
Magnetic Tape	Tape	Up to 1TB	$40 per tape	~240 MB/sec

Figure 82: Backup media

Figure 83: A hard disk next to an SSD

Other factors to consider when creating a backup procedure

Data may need to be recovered from a backup when it is lost as a result of an accident or from the actions of a deliberate threat. Accidents can happen and employees may simply lose or delete data and may not be able to recover it themselves. Deliberate threats such as malware and hacking can also result in deleted or modified data and are discussed in more detail in Chapter 10. Backups should be easily accessible so that the time between the detection of the loss and the recovery of the data is minimal. No matter how good the backup procedure in an organisation, there will often be a version gap between the data that has been lost and the data within the last good backup. For this reason, it is often a good practice to encourage the users of the system to make their own backups, especially if they are making frequent changes.

At least one set of backup media should be kept off-site in the event of a disaster (such as fire or flood). The use of fireproof safes is a good practice, but even though such containers are billed as being 'fireproof', the high temperatures that are reached in a fire can mean that backup media kept within them can be damaged nonetheless.

Multiples of backup media should be produced in the event of the media itself failing. For example, it would be quite dangerous to have one single backup that was being continuously rewritten. Not only would this mean that the backup could potentially fail sooner (due to its high usage), but in the event of failure, there would not be any other backups to use instead. Organisations will often use a set of backup media that they will cycle through (one for each day of the week, for example).

Lastly, a backup procedure needs to include a detailed and well documented restore procedure.

RAID – an alternative to backup?

A Redundant Array of Inexpensive Disks (RAID) can be used to increase the reliability of data and the speed at which it can be accessed. It is not really an alternative to backing up data, but a RAID can be configured so that a real-time backup of data is kept. Different RAID configurations are known as levels and several exist. A RAID 1 configuration utilises at least 2 HDDs and simply copies the contents of one to the others. In the event of one HDD failing, the others have an exact copy. Other RAID configurations spread the data across the HDDs in such a way that if one HDD fails, it can be removed and a new one installed in its place without any loss of data. The data on the other HDDs can be used to 'build' the data that was stored on the failed HDD.

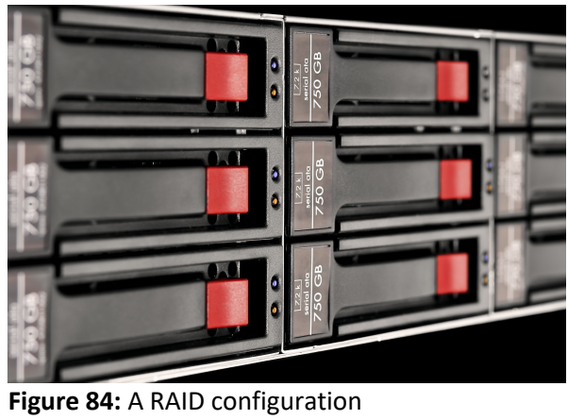


Figure 84: A RAID configuration

Disposal of files

The saying "information is power" has quite a lot of relevance in today's information age. Rarely is information simply disposed of or deleted. Information is more commonly archived in preference to disposing of it as it is often hard to know what information is going to be useful later on. Having said this, there are often data files that are created as temporary files storing various calculations on the way to the production of a final result. It is useful to be able to identify files such as these as their storage can add to the size of a system backup and can make it harder to find important information.

Attributes of files that affect access of data

The size of files, the way files are organised and the storage medium onto which they have been placed all affect the ways in which files can be accessed. This is not a one-way relationship. The need to have files accessed in a certain way will in turn influence the choices that are made about their structure. Files that are used frequently by an organisation need to be placed in a location that is easily accessible and on a storage medium that is both fast and reliable. The size of important and frequently used files can often be large which puts pressure on the network infrastructure as well as the storage device. Less frequently or widely used files can be distributed around a network in locations that are convenient for those using them.

Data structures: The way data is stored to enabled efficient algorithms to be used to optimise program execution time and memory usage. Types of data structures include: arrays, associative arrays, classes, fields, files, hash tables, linked lists, queues, records and stacks.

VCAA VCE Computing Study Design 2020-2023 Glossary

The choice of storage media also has a significant effect on the way that files are accessed. Storage media are often selected based on their capacity, but other attributes that are important are speed of access (latency), reliability, cost and ease of use.

Data structures and how they can be used to store and manipulate data

Data structures provide a way of storing and organising data so that it is easier to access or more efficient to use. There are a wide variety of data structures available to software developers and the choice of which ones to use will be made based on the problem that needs to be solved and the ways in which the data needs to be used.

It is worth noting that we are often not concerned with how data structures are represented in memory, though some structures use more memory than others.

Many of the data structures that are in this course were discussed in Chapter 3. In the section below, we will discuss these data structures and others in the context of how they can be used to store and manipulate data.

Arrays

When a set of data elements is collected under the same name, it is known as an array. Arrays can be created to hold any type of data, the only constraint being that all of the array elements need to be of the same type. Each array element can be separately accessed using an index (or in the case of multi-dimensional arrays, a number of indices).

Number

Index	0	1	2	3
Value	20	15	10	30

Figure 85: An array of integers called 'Number'

Array declaration

Arrays can be declared statically or dynamically.

Static arrays have a defined number of elements while dynamic arrays expand as elements are added to them. Arrays are declared to be of one type only. The benefits of declaring a static array is that its size is known. A software developer can calculate what storage space is available or required, and can declared an array to maximise the use of this space. Loops can then be constructed that iterate through all of the elements within the array easily.

Arrays can also be declared as multi-dimensional arrays. A two-dimensional array (for example), has two indices (which can thought of as an 'x' and 'y' index).

Number

Index	0	1	2	3
0	20	15	10	30
1	65	23	0	-43
2	10	14	5	27

Figure 86: A 2-dimensional array of integers called 'Number'

Array access

Elements in an array are accessed by using the index which points to each individual value. The index of the first element is '0', and increases by increments of 1. This may seem strange, but has evolved from the way that arrays are stored in memory. The first element in the array would be accessed from the first memory location (the one pointed to by the variable). Subsequent elements are accessed via an 'offset' (that is, if the offset is '1', this means that the next element is located at the memory location pointed to by the variable '+1').

For example,

Number[0]

points to the first element in the array. To access the second element of the array, an index of '1' would be used as shown below.

Number[1]

To cycle through all of the elements of an array, a loop can be used that is 1 less than the size of the array.

```
Begin
        Length ← Len(Number)
        For Loop ← 0 to Length - 1
                Display Number(Loop)
        Next Loop
End
```

Figure 87: Cycling through all of the elements of an array

Benefits or shortcomings

Arrays are one of the simplest data structures that are available. They are easy to use and every programming language supports them as a core structure. Many more complex data structures use arrays as their base, so an understanding of how to declare and use arrays is essential for a software developer to be able to make use of these.

The size of an array can make it time consuming to navigate, though there are algorithms (some of which we have already discussed) to make this easier. The fact that one data of one type can be stored is definitely a shortcoming, but the use of 'records' solves this issue.

Knowing the 'bounds' of an array is essential as this is a common source of error in code. It is very easy to 'step' 1 index outside of the bounds of an array (especially at the end). Thorough testing usually picks up errors such as these, but it is also one worth highlighting as it occurs frequently.

Associative arrays

One of the main difficulties with using an array comes when trying to find an element within it. Associative arrays solve this by using a pair of values (one that acts as a unique key and one that stores the value).

"Apple" : "95"
"Banana" : "111"
"Apricot" : "17"
"Kiwi" : "112"
"Lemon" : "17"
"Mango" : "202"
"Orange" : "62"
"Lime" : "20"
"Passionfruit" : "17"

Figure 88: Storing information in an associative array

Let's say that we wanted to record the calories of particular fruits. We could represent this in an associative array like the one below.

Array access

Elements in an associative array can be cycled through just as they can in a normal array. A loop that starts at index 0 and proceeds to the length of the array – 1 can be used to move from the first to the last element. At each index position, the key or the value can be accessed. Doing this doesn't harness the power of an associative array however.

Associative arrays allow a user to access a value via the key. The exact syntax will be programming language dependant, but something like the following is typical.

Name["ID5243"]

Cycling through all of the array elements could be done using code like that shown in Figure 89.

```
Begin
        Length ← Len(Number)
        For Loop ← 0 to Length - 1
                Display Number_Key(Loop)
                Display Number_Value(Loop)
        Next Loop
End
```

Figure 89: Cycling through all of the elements of an associative array

Benefits or shortcomings

In the base form of an associative array, only one value may be stored per key. If a number of values were needing to be stored, this could be accomplished by using duplicate arrays. Some programming languages will allow data structures to be used in the place of the 'value'. In this case, the key can reference an array of values.

Classes

A class is an object that has a number of methods and events associated with it. It can be duplicated as many times as required and used independently of the other instances of itself. Just as a variable can be declared to be of a particular type, an object can be created that is of a particular class and named in a way that allows it to be referenced on its own.

Let's consider an example in which we want to declare a class to store information related to sprinklers in a home sprinkler system. We could represent this as a class like the one below.

Class declaration

Classes consist of variables (within the instance), methods and events. In this example, let's define the class as shown in Figure 90 on the next page.

Using this example, a Sprinkler could be declared using a line similar to the one below:

Sprinkler front_lawn = New Sprinkler( )

```
class Sprinkler

        string Name
        integer Location
        integer Power
        Boolean SprayOn

        public int GetLocation( )
                return Location

        void SprayOn( )
                SprayOn ← True

        void SprayOff( )
                SprayOn ← False

end class
```

Figure 90: Example class definition

The variables within this instance could then be set using lines such as the ones below:

front_lawn.Name = "Front Sprinkler 1"
front_lawn.Location = 1
front_lawn.Power = 5

You will also be able to see some methods that have been defined. These could be called in the following ways:

front_lawn.GetLocation( )
front_lawn.SprayOn( )
front_lawn.SprayOff( )

Benefits or shortcomings

For each problem being solved, there are data structures and techniques that can be used to best suit the data being stored and the processing required. The use of classes can be very beneficial when a large number of complex objects needs to be created and standard variables and functions would be difficult to use. There are many other aspects of using classes that are beyond the scope of this course and text, but in implementing classes, they can be combined and expanded upon in ways that make them extremely powerful. If you wish to investigate these further, read up on the concepts of encapsulation, inheritance, polymorphism and abstraction.

On the negative side, classes can be tricky to define initially.

Records and fields

Easier to implement than classes, but with similar functionality on a base level, are records and fields.

A record is a structure that can be used to group together variables for a particular purpose. Records are similar to arrays except that where an array usually contains elements all of the same variable type, the fields within a record may be of different types and sizes.

Accessing a record or field

Let's consider an example where we are storing information related to the items being sold in a store.

```
item(index).id ← "0011"
item(index).name ← "Screwdriver"
item(index).brand ← "Champion"
item(index).model ← "MX-52"
item(index).stock ← 12
item(index).location ← "3B"
item(index).price ← 10.99
```

Figure 91: An example record data structure

Benefits or shortcomings

Records have some specific advantages over using 'plain' one dimensional arrays. If data like that shown in the example were to be represented in arrays (one for each field), each one would be disconnected from the others and would not make a lot of sense on its own. For example, if the price were required for the item with the 'id' of '0012', the index of this item would need to be found in the 'id' array and then this index would need to be used in the 'price' array. The main difficulty with such an arrangement comes when the information

needs to be sorted in some way. Sorting a number of parallel arrays (and keeping them in sync with each other) is a difficult task.

Files

While we have been dealing with ways in which data can be represented in memory, it is also important to consider how data will be stored. Storing data on secondary storage devices (such as hard drives), requires the use of a file. A file is a way to organize stored data for access at a later time.

In Chapter 3, we discussed the use of TXT, CSV and XML file formats. Each file format has its own benefits, but files will all generally be accessed in similar ways in a software solution. An example from an XML file is shown below:

```
<stock item>
        <id>0002</id>
        <type>Fish</type>
        <name>Comet Goldfish</name>
        <quantity>15</quantity>
        <cost>$12.23</cost>
</stock item>
<stock item>
        <id>0003</id>
        <type>Plant</type>
        <name>Kelp</name>
        <quantity>20</quantity>
        <cost>$0.43</cost>
</stock item>
```

Figure 92: An example section of an XML file

Benefits or shortcomings

When accessing a file within a program, it is important to control its access carefully. Files need to be opened for use and then closed after the program has finished reading from them and writing to them. Files should be kept open only for the amount of time that is required to read or write to them. There is a danger that if the program crashes and it has files that are currently open, that these files will be corrupted. As data is such an important commodity, protecting it should be at the forefront of any solution design. In addition to this, multiple users will often access files simultaneously and will have different permissions to access the file.

Choosing between TXT, CSV and XML files

When choosing to use a file to store data, the format will be largely dictated by the amount of data that will be stored and the ways in which it will be accessed. For example, if a software solution needs to store some default values to be used in the software solution, a TXT file will suffice.

When a software solution needs to store array values or fields associated with records, a CSV file can be a good solution. The thing to be aware of when choosing a CSV file format over and XML format, is that the number of fields needs to be constant for each record. In addition, when writing to or reading from a CSV file, the order of the fields needs to be hard coded into the software solution.

In contrast to this, an XML file adds flexibility of format and structure. The tags within an XML file indicate the name of the field. Not only this, but they can be in a different order in each record, and records can contain different fields. An XML file is very easy to read and understand in isolation, and a software developer could easily utilise it without having to read any documentation to know what it contains.

Hash tables

A hash table is essentially a way of implementing an associative array. While an associative array of unlimited size can be used to store the occurrences or the values associated with key variables, a hash table uses a hashing algorithm to store these values more efficiently.

The elements within a hash table data structure consist of a key and a value (just as they do in an associative array). A hash function will work out the index in an array where the key:value pair will be stored. In the

case of the array index being used already, the key:value pair is attached in a process known as 'chaining'. In effect, each element of the hash table should be an array or list in its own right.

An example of a hash table is shown below:

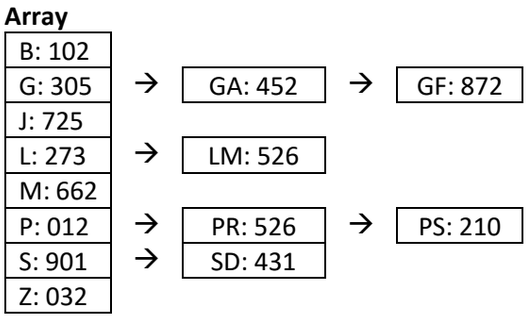


Figure 93: An example of a hash table

Benefits or shortcomings

Hash tables (like associative arrays) make it quick and easy to locate specific elements with the array. While it is more ideal if the element being searched for is one that exists, it will be easy enough to find one that does not exist, by hashing to the index where it should be located, and following the chain till the end.

If a poor hashing algorithm is used, a hash table could potentially have a large number of elements in a small number of chains and in effect, become just like a one-dimensional array. In this case, searching for elements would become very inefficient. However, a hashing algorithm could be assessed to ensure that it is distributing elements evenly throughout the array, and if it is performing in this way, the benefits of using a hash table will be easy to see.

Stacks

A stack is a data structure that works on the principle of First In - Last Out (FILO) (or Last In – First Out depending on your perspective!). A stack is a one-dimensional array of values that are accessed in a particular way.

There are only two ways in which a stack data structure can be accessed. Data can be placed on 'top' of the stack. This function is called the 'push' function. Data that is taken off the stack can only be taken off the 'top' and this function is called the 'pop' function. Diagrammatically, a stack could be represented as shown in Figure 94.

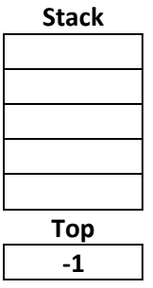


Figure 94: A diagrammatic representation of a stack

Note however that a stack is just a one-dimensional array. Drawing it vertically does help to visualize what is going on within the stack, as the process of placing items on top of the stack and taking items off is the same as what would happen with a physical stack of objects.

In order to operate a stack, two things about the stack must be known. We must keep track of the position (or index) of the 'top' of the stack. We must also know how large the stack it, just in case we fill it to capacity.

Consider the stack from Figure 94. If we were to perform the following 'push' and 'pop' commands, this would add items to the stack and remove them. Note that as items are added to the stack, the 'top' variable is changed to the index of the top element.

Push(10)
Pop
Push(3)
Push(22)
Push(74)
Pop
Push(17)

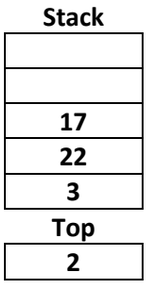


Figure 95: The stack after some elements have been added

Benefits or shortcomings

Stacks are useful data structures as they operate in a First In - Last Out way. This is particularly useful in executing code and backtracking through calls to modules and functions once they have been completed. In fact, you may have encountered a 'stack overflow' error if you have had a circular function call in some code that you have written.

While easy to implement, stacks are limited by their size. A software developer will aim to make the size of the array to be used for the stack big enough so that it will cater for the requirements of the problem, but not so big that it is using up a large amount of memory unnecessarily.

Many languages have stack data structures available natively, so that software developers don't have to code their own 'push' and 'pop' routines. In these cases, it is only necessary to declare a new stack variable and then use the available 'push' and 'pop' methods to place items on to the stack or remove them.

Queues

A queue data structure can be implemented in a very similar way to a stack except that it works as a First In - First Out (FIFO) structure.

Items in a queue are added to the back and removed from the front. For this reason, it is necessary to keep track of where both the front and the back of the queue are positioned in the array. Adding an item to the back of the queue is referred to by the term 'enqueue' while removing an item from a queue is called 'dequeue'.

Let's work through an example so that you can see how a queue works. The following commands will be executed on the queue shown in Figure 96 below.

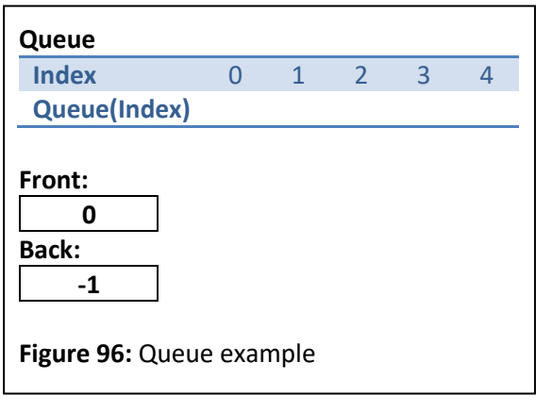


Figure 96: Queue example

enQueue(20)
enQueue(40)
deQueue
enQueue(35)
deQueue
enQueue(15)
enQueue(10)

The 'front' variable points to the front of the queue while 'back' points to the back of the queue (in this case 'back' is set to '-1' to indicate that the queue is empty).

As items are added, they are added to the back of the queue. As items are removed, they are removed from the front of the queue.

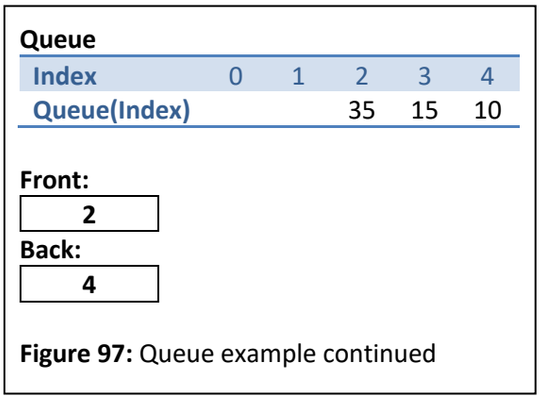


Figure 97: Queue example continued

Benefits or shortcomings

Queues can be useful data structures to implement as many real world situations operate in a First In – First Out fashion. A simple printer queue is a classic example of a queue data structure in action.

In practice, the process of adding elements to the queue and removing them has the side effect of moving the queue forwards through the array. As you can see in Figure 97, the queue as it stands is located in array elements 2 to 4. If another element were to be added to the queue as it stands, the algorithm would fail as the queue is full.

There are two ways that this can be prevented in practice. The first way is to move the queue back down the array when it gets close to the end. This isn't very efficient as it adds to the processing time. The second way is to implement what is known as a 'circular queue'. While beyond the scope of this course and text, a circular queue connects up the beginning and end of the array logically speaking. Items added to the end of the queue may be located at the beginning of the array if the queue has moved beyond the end.

Like a stack, a queue still needs to have enough space allocated to it to cater for the maximum amount of data that is likely to be stored within it.

Linked lists

A linked list is a data structure that maintains itself in sorted order at all times. One of the main disadvantages of using arrays to store data is that it is very difficult to maintain them in sorted order. There are sorting algorithms that can be used to rearrange the data within the array (two of which we examined in Chapter 3). However, sorting algorithms take time to execute. By using a linked list, items are placed in their correct order straight away.

To create a linked list requires two parallel arrays: one to store the required information and one to store the links which will point to the next array element. It is also necessary to store the index of the first item in the list.

A small linked list might look like the example below:

Linked list

Index	0	1	2	3	4
List(Index)	30	40	35	20	10
Link(Index)	2	-1	1	0	3

First:

4

Figure 98: Linked list example

In the example above, you can read the list of numbers in order by beginning at the index pointed to by 'First' and proceeding to follow the indexes associated with each array element. If you do this, the order will be:

4, 3, 1, 2, 1, -1

The '-1' indicates that the end of the list has been reached.

Benefits or shortcomings

The example shown here is obviously a very simple one and has been used to illustrate how a linked list works. In practice, linked lists are much larger than this and also utilise a variable pointer called 'free' which indicates the next free array element. While there is some overhead in relation to maintaining the links, this is a small price to pay compared to sorting an array from scratch. One of the main advantages in using a linked list data structure is that elements are added in order. This does require that the algorithm locate where in the order the item belongs. In contrast to this, removing elements is a simple matter of changing the links so that they skip over the removed element. Over time, this can lead to 'holes' in the linked list, so an advanced method of preventing this is to also maintain a linked list of the free array elements.

Context Questions

1. Why is it important to use a file naming convention?
2. Why is it important for two software packages that are going to be exchanging data via text files, to use the same delimiter?
3. List the three types of backup procedure.
4. Company A is using a differential backup procedure and company B is using an incremental backup procedure. Both companies have the need to restore their backups due to a catastrophic loss of data. Which company will have their backup restored quicker and why?
5. Why is it important to have at least one backup stored off-site?
6. How often should a backup be performed?

Applying the Concepts

- Investigate and then design a backup system for your home network and present this solution to your class for feedback.

Key Skills Checklist

At the conclusion of this chapter, you should be able to address the following key skills. Mark each off as you can achieve them.

Generate new design ideas using one of several methods	☐
List and describe efficiency measures of a software solution	☐
List and describe effectiveness measures of a software solution	☐
Describe factors that influence the design of solutions	☐
Discuss what makes efficient and effective user interfaces	☐

Sample Examination Questions

The following sample examination questions can be attempted to test your knowledge of the content of this chapter.

Question 1
Which of the following would be a good example of the use of a file naming convention?

- **A.** Document1
- **B.** File050613
- **C.** Smith_AuditData_Expenses_070416
- **D.** Smithfamilyauditexpenses56

Question 2
Turtle Enterprises performs a full backup of its' data each Friday after work hours (on an automated process) at 11pm. For Monday to Thursday, an incremental backup is done each day at the same time in the evening. On Thursday morning, there is a hard drive failure and all data is lost. Which backups will need to be restored to reinstate this data?

- **A.** Wednesday
- **B.** Friday and Wednesday
- **C.** Friday, Monday, Tuesday, Wednesday
- **D.** Friday, Monday, Tuesday, Wednesday, Thursday

Question 3
The data structure Length[Index] is most likely to be:

- **A.** A one-dimensional array
- **B.** A two-dimensional array
- **C.** A reference to a hash table
- **D.** An integer

Question 4
What is the difference between full backup, differential backup and incremental backup?

_____

_____

_____

3 marks

Question 5
A company has decided to change from performing a daily incremental backup to performing a single full backup once per week. List **two** disadvantages of such a move.

Disadvantage 1: _____

Disadvantage 2: _____

2 marks

Question 6
Khalil is considering different data structures that he can use in the creation of a software solution he is writing, that will store all of the information related to a client.

a. What is a data structure?

_____

_____

1 mark

b. Khalil is deciding between using a record data structure and a simple 1D array. Which of these would be the better choice? Give **two** reasons to support your choice.

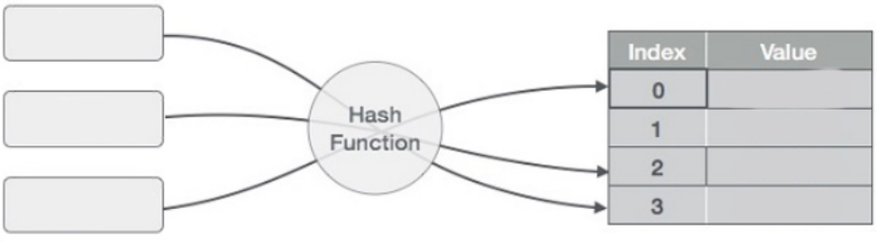
Choice: _____

Reason 1: _____

Reason 2: _____

2 marks

Question 7
The diagram below shows how a hash table works conceptually.



a. Explain what a hash function does.

_____

1 mark

b. What is the main advantage that a hash table has over a one-dimensional array?

_____

_____

1 mark

Sample Examination Answers

Question 1
Answer: C

Questions such as this are always 'best possible answer'. Option C in this case is quite good as it contains a person's name, a reference to what the file actually is as well as a date.

Question 2
Answer: C

They will need to reinstate first the full backup, followed by every incremental backup in turn.
Question 3
Answer: A

Question 4
Full backup: backup all of the data every time.
Differential backup: only record the changes to the data compared to the last full backup.
Incremental backup: only records the changes since the last incremental backup.

Question 5
Disadvantage 1: A full backup takes longer to perform than an incremental backup (discounting the first day).
Disadvantage 2: If the server fails the day before the full backup is due to take place, then potentially 6 days of data could be lost (as opposed to a maximum of 1 day for an incremental backup).

Question 6
A data structure is a way of organizing data in a computer so that it can be used effectively.
Choice: record
Reason 1: Each of the fields associated with the client's record can be kept together.
Reason 2: A 1D array would not be able to store more that one data type.

Question 7
A hash function maps a value to an index based on a rule.
It is quicker to locate values or determine if they are present than checking all of the values in a one-dimensional array.

Chapter 8
Testing and evaluating

The chapter covers Unit 4: Area of Study 1 key knowledge:

- *Digital systems*
  - *KK3.5 Characteristics of efficient and effective solutions*
  - *KK3.8 Techniques for testing the usability of solutions and forms of documenting test results*
  - *KK3.9 Techniques for recording the progress of projects, including adjustments to tasks and timeframes, annotations and logs*
  - *KK3.10 Factors that influence the effectiveness of development models*
  - *KK3.11 Strategies for evaluating the efficiency and effectiveness of software solutions and assessing project plans*

Key terms: Efficiency, effectiveness, testing, test data, testing table, project plan, testing, useability, large file sizes, benchmarking, live data, documenting testing, evaluation, strategies, acceptance testing, quality assurance.

Study roadmap

	Unit 3		Unit 4	
Ch	Area of study 1 – Programming	Area of study 2 – Analysis and Design	Area of study 1 – Development and Evaluation	Area of study 2 - Cybersecurity: software security
1	✓			
2	✓			
3	✓			
4		✓		
5		✓		
6		✓		
7			✓	
8			✓	
9				
10				

Characteristics of efficient and effective solutions

In Chapter 6, we discussed various measures of efficiency and effectiveness and how attention to these can affect the design stage. We will now revisit these measures and examine them in light of development and testing.

Efficiency measures

Speed of processing

When examining the processing speed of a software solution, there are a number of factors that play a role. Entering data into a software solution is certainly part of this. There may not be much processing taking place at this time, but interfaces may be working to dynamically populate fields and delays in doing so will be noticeable. When output is produced, is the software solution able to generate this output in a timely manner? Switching from one view in the interface to another can also be dependent on processing speed.

How can we improve speed of processing?

Speed of processing can be improved in a number of ways. The most obvious one may be to improve the hardware (processor, available

memory), but this may be out of the control of a software developer. Let's examine some other ways that speed of processing can be improved.

- Some data structures will be quicker to use than others. An assessment of the data structures that have been chosen in the context of how well they are responding, is a worthwhile exercise.
- Examine the solution's use of memory. Are data structures sized reasonably? Are there variables that have been declared that are not being used or ones that could be reused?
- Examine how well the key sections of code perform their functions. For example, there are a large number of sorting algorithms available and each has their strengths and weaknesses. Implementing a quick sort might seem like a good idea, but arrays that are in reverse order or have many duplicates can cause a quick sort algorithm to perform poorly. Likewise, the placement of the pivot can affect how well as quick sort works for a set of data.
- Screen operations are slow in comparison to those taking place in memory. In cases where a software solution is making a large number of changes to the GUI, it can be quicker to write changes to screen RAM directly and have the whole interface displayed at once with the next clock cycle of the display.
- The biggest delay in speed of processing can occur when reading or writing to storage media. As before, this may be out of the control of the software developer, but the way that data is read or written could be changed and the number of times this is done, minimised.

Functionality

The functionality of a software solution can have a profound effect on the efficiency. How well are the functions within the software solution designed? Do they address the tasks for which they have been designed with the minimal amount of interaction from the user?

How can we improve functionality?

Methods of testing a software solution were discussed in Chapter 3, but improving functionality is different from testing in that the main focus is on the ease of operation.

- A usability test is a useful tool that can be implemented to improve functionality. In constructing a usability test, a software developer will write out a list of tasks and steps typical of the normal operation of the software solution. They will then ask others to use the software solution following these steps and give them feedback on how well they were able to execute these. When writing a software solution, it is sometimes quite easy to become so familiar with its operation, that aspects that are not logical to a first-time user will be difficult to spot. This process helps to highlight these. You can read more about usability testing later in this chapter.
- Are there variables or data structures that can be automatically populated either completely or partially? Constructing code to do this may take extra work but may remove steps that the user needs to complete.

Cost of file manipulation

As mentioned in the speed of processing section, accessing files can have a significant effect on the efficiency of a software solution.

How can we improve the cost of file manipulation?

Any file operations will be relatively slow. The challenge can be to structure these in a way that allows them to have minimal impact. As discussed previously, changing the storage media can be beneficial but may be out of the control of the software developer.

- File reads and writes can be held till there is an opportunity to do them all at the same time. One way this can be accomplished is to construct files in RAM first and then write to the storage media

when the software solution is being closed. The downside of this is that there is increased risk of losing data.

Effectiveness measures

Completeness

Completeness is the measure of how well the information that is being produced meets the purposes for which it is intended. While it is in some ways a subjective measure based on the task that is being performed and the user's needs, there are ways that a software developer can work to improve this area.

How can we improve the completeness of the information?

- Throughout the design and development stages, a software developer can show the client what information is being produced and gain feedback on the appropriateness of it and make adjustments as required.
- In designing a GUI, it is often tempting to present all of the available information to the user at once. Doing so however, can lead to issues with attractiveness and clarity. Instead of limiting the information, a software developer could consider having some of the information accessible on different screens or hidden from the default view.

Readability or clarity

Readability or clarity is simply the measure of how easy it is to read and understand the output that is produced. If a user struggles to read or understand the information that is being presented, the effectiveness with which they are able to use the software solution rapidly deteriorates.

How can we improve the readability or clarity?

- As a general rule, clarity can be improved by using simpler language, removing the use of jargon (where possible) and using shorter sentences. Online tools such as the Hemingway App will take text input and give feedback based on the education level required to read the text and feedback on what changes can be made.
- Allowing the user to change the font size and interface colours can aid in improving the readability of text.
- The choice of font is important as some fonts are well suited to headings while others are better suited to descriptive passages. In addition to this, numbers in some fonts are easier to read than others.

Attractiveness

It can be difficult to relate the attractiveness of a software solution to efficiency. Colour scheme, fonts, consistency of objects in the GUI and how the software's functions are accessed, all have an influence on how effective the solution is and can all be considered to belong in this area.

How can we improve the attractiveness with a view to increased effectiveness?

- Just as with the area of clarity, by improving the look and feel of the software solution, it will become easier to read, use and understand.
- An attractive interface will make it more likely that a user will be inclined to use it.
- An attractive interface may also lead the user through the process rather than challenge them to work out what to do next.

Accuracy

Accuracy would seem to be an area that is easy to access and improve upon. When assessing accuracy, a software developer will be testing the solution to see that the information being produced is correct. In addition to this, they will be checking to ensure that inputs are designed in such a way as to prevent inaccuracies being introduced into the system.

How can we improve accuracy?

- Testing tables can be used to show how accurate the information being produced is and compare this to known results. In cases where the differences are larger than what is acceptable, the calculations can be adjusted to ensure that this is improved.
- In some cases, the use of particular variable data types will have an effect on the accuracy of a software solution. Ensuring that the data types being used match the data that is being stored should eliminate this.
- The use of strong validation techniques can prevent incorrect or inaccurate data being introduced into the software solution.

Accessibility

By definition, the accessibility of a software solution is the practice of allowing it to be used by the greatest range of users without placing barriers in their way. A software solution that is not accessible for some, will mean that it is being used ineffectively (or not at all) by those users and this will have an overall negative effect on the effectiveness of the software as a whole.

How can we improve accessibility?

- Functions and navigation should be clearly designed in a way that indicates their function. On screen help and tips that appear when the user hovers over objects can aid in this.
- Software solutions can be designed with the specific client in mind. The client's experience, age and their role can all be factors worth considering.
- Accessibility is also concerned with ensuring that those with disabilities are catered for. This can mean designing interfaces that accept input from a variety of sources, including text to speech functionality, catering for those with colour blindness and allowing the user to change the colour scheme, fonts and font sizes.

Timeliness

Does the software solution produce information within a time frame that is relevant and useful? A software solution that doesn't do this, risks being completely ineffective.

How can we improve timeliness?

- The first step in improving timeliness is coming to an understanding of the time frames involved in each particular instance. When is data available and what are the expectations from users in regard to when information will be required? Once this is understood, the software solution needs to be designed to work within this time frame.
- In cases where timeliness is still an issue, a software developer should examine the ways that data is obtained to see if there are methods that can automate or improve the process.
- Processing times may have a lesser effect on timeliness but may be something that needs to be considered. If the information that is being produced needs to be available within seconds, then the algorithms, data structures and calculations that are being implemented should be examined for ways in which they can be refined. Just as with 'cost of file manipulation' discussion in the efficiency section, delaying or streamlining reads or writes to storage media can have a significant effect on the processing time required.

Communication of message

In considering how effectively the software solution is presenting the information that it is producing, there are a number of measures that can be used. For example, is the information easy to read and interpret or is it being presented in a way that is acting as a barrier to this. The software solution may have all of the correct information on hand, but just may not be presenting it in way that is very useful or accessible.

How can we improve the communication of message?

- In performing usability tests, feedback can be gained from users in regards to how easily they were able to locate the required information and ways in which they feel this acquisition process can be improved.
- The use of familiar icons, symbols and interface features will make it easier for users to determine how to use and locate the functions they need.

Relevance

Is the information that is being produced by the software solution relevant to its purpose? Are there times when the solution gives information that is not strictly targeted to the person using the software?

How can we improve relevance?

- By including specific levels of user access that restrict the amount and type of information that is displayed, users can be presented with information that is targeted to their needs.
- Provide options to users that allow them to select extra information if they require it.

Usability

Well-designed software solutions should have interfaces that allow the user to perform the tasks they need to accomplish without getting in the way. There is a fine balance between all of these measures and sometimes factors such as attractiveness and accessibility can be at odds with usability. The important thing is that the processing that is required should remain the central focus as opposed to how the solution looks.

How can we improve usability?

- One of the best ways to improve usability is to conduct a usability test. A usability test asks users to perform typical tasks and then asks them for their feedback on the aspects of the solution that they felt performed well or could otherwise be improved.

Testing

Testing a coded solution for errors

When a program is first being tested by the creators of the program, it is said to be in 'alpha' testing. 'Alpha' testing is also known as 'white-box' testing. Once a program has passed 'alpha' testing, it is released to select members of the public for further testing known as 'beta' testing. 'Beta' testing is also known as 'black-box' testing. The main difference between 'white-box' testing and 'black-box' testing, is that those who are carrying out the 'black-box' testing do not have access to the code and are concentrating on features like function, usability and appearance. Once a specified time of 'beta' testing has been completed (and the bugs corrected), the software package can be released officially.

The process of testing a program is one in which it is determined whether the program will produce the expected solutions under a variety of conditions. This not only implies that it will produce the correct output for a set of valid inputs, but that it will reject input that is not in the correct form (that is, validate it as completely as possible).

To perform a thorough test of a software package, a table like the one below is usually produced.

Table of tests

Item Tested	Test Data	Expected Result	Actual Result

Figure 99: Testing table

Tests are then methodically carried out on all aspects of the program's operation. Where the program does not perform as expected, notes are made in the 'actual result' column so that changes can be made or bugs fixed.

Selecting test data

When testing a program using a testing table, it is very important to select a range of test data that ensures that the program operates as expected in all situations. Firstly, pick typical values for which you can easily calculate or predict the results they will produce. You should also pick values at the extremes of the allowable data input range to ensure that validation rules are operating correctly. Next, select values that are outside of the allowable data range to ensure that the program does not allow this data to be accepted and displays an appropriate message to that effect. Lastly, it is always good to include nonsense test data that will ensure that the validation procedures are robust and the program is stable.

'Boundary value analysis' is the name given to the testing technique involved in testing values both inside and outside of the allowable inputs for a software solution. As the bulk of logic errors occur at the boundary of valid inputs, this technique involves checking values at the boundary, just inside the boundary and just outside the boundary.

Testing the usability of software solutions

The original design specifications need to be checked against the completed software solution to ensure that all components are in place and operating correctly. Factors of interest at this stage will be response times, how the system copes with transactions involving very large file sizes, coping with transactions which have a mixture of data types and how modules interface with each other.

Usability testing

While 'Alpha' testing is carried out by the software developers and is concerned with debugging the code and ensuring that it is operating correctly, usability testing is carried out by some of the intended users of the software solution.

Usability testing could be considered to be a type of 'Beta' testing, though it is more focused.

In carrying out usability testing, a select group of users are given access to the current version of the code and asked to perform standard tasks (usually following a specific script). In carrying out this testing, users give feedback on how easy it was to perform tasks, how easy the interface was to understand, any errors that take place and potential improvement ideas they may have. This feedback is given back to the developers who will compile them and create an action plan to address them.

Large file sizes

Testing how the system copes with large file sizes is important as it may be something the system needs to deal with on a regular basis. In addition to this, it is important to discover what the limits of the system are, even if the files used at this stage are far in excess of what is expected. Over the course of a few years, the volume of data may change dramatically and knowledge concerning the limits of the system will be valuable in planning for the next system upgrade.

Benchmarking

The process known as benchmarking examines and measures the tasks within an organisation and compares them to best practice. It is a large field in which much has been written, but we will only make a small mention of it in this context. It is more commonly performed in the analysis stage of the PSM as a means of determining problems within the system. At this stage, benchmarking is a means of documenting the capabilities of the new

software solution and comparing these with the design specifications.

Live Data

'Live data' is a term used to describe data that flows through the system in real time or while the system is in normal operation. Testing a system using live data is a very important step as it has possibly not been exposed to the rigors of its intended environment prior to this. The software developers have certainly thoroughly tested the system using simulated data, but this is not the same as live data and does not place the same demands on the system in terms of required performance.

How will the system cope when a large number of users are simultaneously executing different processes or entering data of different types? How well does the system validate user input and prevent problems from occurring in a live environment? Are there unacceptable delays in response times when employees are using the system at the same time or at certain times of day?

Documenting test results

The process of documenting test results is an important one. While it is easy to see that conducting thorough testing is a vital step, it can be hard to see why it would be important to finish off the testing stage by documenting all that has been carried out.

Documenting testing creates a record of what has been carried out, how successful it was and what measures were taken to correct any errors that were found. This information is useful for management in reporting and being accountable to clients or the legal team.

Some organisations use a database system, documenting the test results on a report template. One alternative is to purchase a commercial product that tracks and produces reports on software testing. With these approaches, you can automate reports to monitor test results and their progress. Whatever method is chosen, it is important that members of the project team can access the documentation system easily.

However an organisation decides to track and document testing, it is important that software developers and testers are familiar with the required format of this documentation and that it is consistent.

Items typically included would be:

- Tester's name and their department
- Software package and version (or module name)
- Date and time the tests were performed
- Full description of the results
- Resolution of any problems

Evaluating the solution

It may seem strange to be considering how the software solution will be evaluated at this stage in our journey, but in fact, an appreciation of when and how the solution will be evaluated is a very important consideration. An awareness of these criteria prior to the completion of the software development process will help to keep the development on track. When deciding on criteria for evaluating how successful a software solution is, first and foremost should be the consideration of whether the solution meets the Software Requirements Specification (SRS).

Data can be gathered using some of the following methods: surveys and questionnaires, interviews and focus groups, as well as direct observation. As these have been discussed in a previous chapter, they will not be described again here.

So what form do assessment criteria take when used to evaluate the software solution? Often they can be simply expressed as a series of questions. For example, does it cost less to perform the same task in the new system compared to the old? Are customers satisfied with the service that they are receiving? All criteria created for an evaluation should relate

An example evaluation strategy		
Time Frame	**Description**	**Efficiency and Effectiveness Measures**
Prior to signing off on the finished software solution	Acceptance testing by the users of the solution (more detail on the acceptance testing process can be found on the next page)	Efficiency • Is the software solution easy to use and understand? • Is the software solution able to be used to produce results in a reasonable amount of time? Effectiveness • Are the correct results produced when known values are entered?
Immediately after the software solution has been installed	Network or technical staff ensure that the software solution is operational and ready to be used	Effectiveness • Is the software solution fully operational?
3 – 6 months after the introduction of the software	Feedback on the software solution gathered from the following groups and using the methods below: Employees via: • A survey • Interviews with select staff in each department Management via: • Interviews System support via: • System error logs • Reports on help desk requests Customers and clients via: • Feedback forms	Efficiency • Is the software solution as easy to use and understand as it was initially? • Does the software solution continue to produce results in a reasonable amount of time now that all users are using the solution at once? • How much time has the software solution been down due to maintenance or errors? Effectiveness • Is the correct information being produced by the software solution?

Figure 100: An example evaluation strategy

to the identified system goals, aims and objectives.

When evaluating a solution, it can useful to frame the criteria around measures of effectiveness and efficiency. As stated previously, effectiveness can be measured by examining whether the goals of the system have been met. Effectiveness is a measure of how well the solution performs its assigned tasks. Often the best way to frame criteria is as a question that can be posed in relation to the solution. Some examples are listed below.

Efficiency criteria

- Does the solution complete all processing tasks in an acceptable amount of time?
- Is the software solution easy to use and functional?

Effectiveness criteria

- Is the information that is being produced complete, relevant and accurate?
- Is the output readable and the message well communicated?
- Is the user interface and output easy to understand and attractive?
- Are the on-screen instructions clear?
- Is the information produced within a time frame that ensures that it is still relevant and useful?
- Is the software solution easy to use?

Strategies for evaluating the quality of solutions

Evaluation of a software solution usually takes place after it has been in operation for a while. Evaluating a software solution is a complex task and strategies need to be formulated to ensure this task is a manageable one.

In Chapter 6, we discussed some of the efficiency and effectiveness measures that are used to define solutions. The techniques known as acceptance testing and quality assurance are ways to compare these measures with the initial goals of the software solution as outlined in the SRS. However, before these techniques can be used, strategies must be devised to acquire the necessary data.

It is no coincidence that the first activity listed under evaluation in the PSM is determining a strategy. A strategy is a plan of action designed to achieve a particular goal. In this context, the goal is the evaluation of the solution.

A strategy to evaluate the quality of a solution is a more complex activity than surveying the users of a system or checking system logs. An evaluation strategy should contain an explanation of the timeframes that are involved, the criteria that will be used to conduct the evaluation, the users that will be affected and the techniques that will be used to gather data. Gathering evaluation data can be done in the same ways as data was gathered during the analysis phase.

Acceptance testing and quality assurance

Quality assurance is the process by which software developers ensure a new software solution is operating correctly. They need to ensure that its function meets or exceeds the expectations of the designers and the organisation. This is shortly followed by acceptance testing.

Acceptance testing is so named because the employees and/or management of the organisation are involved in ensuring everything is operating as it should. Representatives of the organisation are given a chance to test the system for themselves and give feedback to the development team on any problems or faulty processes. Specification creep (also known as scope creep) must be dealt with at this point because the testing is intended to show that the original design goals have been met. Where the organisation has been adding to the goals during development, the test data must be extended to show that the extra specifications have been met. This of

Criteria and techniques for acceptance testing	
Does the software solution perform the tasks set out in the SRS?	Identify the main tasks required of the software solution as identified in the SRS and place these in a testing table – where the actual result can be compared to the expected result.
Is the software solution effective (producing the correct / expected results)?	Compare information produced using the previous information system (if this is available) to that produced using the software solution. Manually verify the information that is produced using the software solution.
Is the software solution efficient in terms of its use (time to produce results, ease of accessing functions within in)?	Compare the time taken to perform tasks using the software solution to the time taken using the previous system (or what is expected of the software solution).
Is the software solution user interface intuitive?	Select a number of different users within the organisation and ask them to provide feedback on the location of functions, ease of understanding, consistency of navigation, etc.

Figure 101: Criteria and techniques for acceptance testing

course has a significant effect on the effectiveness of the project plan that was created at the beginning of the software development process.

Factors that influence the effectiveness of project plans

Lots of factors can have an influence on the effectiveness of a project plan. While a project is in progress, it is not possible to fully anticipate all of the problems or interruptions that will occur.

The following can all be factors in limiting the effectiveness of a project plan.

Clear scope

Has the original scope of the project been clear enough and has it been correctly translated into the specifications and goals for the solution.

Specification creep

Specification creep occurs when the clients of a software solution add features to the requirements even though the development has already begun. Creep is more likely in projects that have long timelines and is often caused by changes in personnel, the marketplace or to compete with the latest innovations of competitors.

Changes in staff

Changes in staffing while the project is underway can have a significant effect, especially if those staff are the project leads or key members of the developer's team. Staff taking or returning from leave can also have an effect as there needs to be time allocated to handover in both cases.

Communication issues

Too many meetings can slow a project down while not having enough meetings can mean that team members can potentially go off on tangents or be idle.

Inadequate time for testing

If inadequate time is allocated for testing modules at different stages in the development, problems could develop in later stages that require these modules to be fixed or modified.

Budget constraints

If modules take longer to code that expected, the most obvious effect on the project plan is that milestones can be pushed out. However, a hidden effect is on the budget of the project as a whole. Longer development time could mean that more money is spent on coding than was allocated and this may in turn lead to other aspects of the project having to be cut down.

Dependent software

No coded software solution works in isolation. There are always other software packages that it needs to integrate with on some level. Sometimes a constraint like this is hard to quantify in terms of the time and effort that will be needed. Once the process is started, unforeseen problems can crop up. There is also the unknown factor of dealing with the developers of these packages. They may be very willing to work with you or may be busy or disorganised.

Technology changes

Technology changes can happen while a software development is in progress. This could be new hardware releases, newer versions of operating systems or software drivers or newer versions of software packages that the coded solution needs to integrate with. Once again, changes like this are very hard to predict and can lead to lengthy delays in a project as the implications are assessed and changes made.

Strategies for evaluating the efficiency and effectiveness of project plans

At the beginning of a software development process, a project plan is created to support an organised approach to the problem-solving methodology. This project plan (as described in Chapter 4) consists of a list of all of the tasks that need to be completed, the team members that will be involved in the project and what their responsibilities will be as well as the schedules that the project needs to adhere to.

As the project progresses, this project plan is examined frequently to ensure that timelines are being kept to. A useful strategy to employ is to make annotations to the project plan as they happen.

Adjustments to timeframes and tasks

Projects almost never finish in the manner that was expected of them and recorded in the initial plan. There will be variations in times, tasks, schedules and available resources. A good project manager will adjust these aspects of the project as they occur, so that potential problems or future changes can be catered for.

A project plan that is stored in the cloud and available to key personnel, allows all involved in the project to see these changes and to keep an eye on upcoming tasks and milestones. While maintaining a dynamic plan such as this has obvious benefits, it is also beneficial to save versions of the plan and to annotate changes.

As a project progresses, a good project manager will regularly examine the plan and adjust tasks for the times that they are completed or estimated to be finished. This will in turn have an effect on the overall timeframes within the project and may mean that some resource bookings need to be modified (outside contractors, system testing times).

A good project manager will also make annotations to tasks as changes are made or additions catered for. They will keep a comprehensive log of changes so that when they need to make reports to their superiors or the organisation, they are able to explain the variations to the initial plan that have taken place and the impact of these changes.

Annotations should changes such as the following:

- The actual duration of tasks compared to their expected duration.
- The achievement of milestones.
- What modifications to the project plan were performed and notes explaining why this was the case.

If this is done in a methodical manner, at the conclusion of the project, the original project plan can be examined and compared to what transpired over the course of the development. This will help to inform future projects as well as explain delays or changes that have occurred to the client.

Evaluating the effectiveness of development models

One of the very first decisions that software developers face when beginning the development of a software solution is, which development model will be most suited to the project. In Chapter 4, we discussed three of these development models: waterfall, agile and spiral. Each has its advantages and disadvantages as well as situations to which they are well suited. However, software development is not an exact science and it is not always easy to see which is the best choice or indeed foresee circumstances that might make one model favored over another. The ability to evaluate the effectiveness of these development models and the factors that have influenced this, gives valuable insight into each.

Factors influencing the effectiveness of the waterfall model

As discussed previously, in implementing a true waterfall development model, each stage of the PSM generally finishes before the next one can begin. This transition is not seamless; there are typically 'gates' between the stages. For example, at the conclusion of the analysis stage, the client would need to sign off on the SRS. Once this has happened, and only then, would the design stage be initiated.

There are certainly both good and bad aspects to using the waterfall development model and we have discussed these already. At this point in our discussion, let's consider the factors that can occur in a project utilising the waterfall model.

- While developers and the client agree on what will be delivered early in the PSM, there is some argument to be made that this is too early. Clients are sometimes intimidated by having to provide such specific details, so early in the project. They are not always able to visualise the finished product or possibly, what their exact needs are. A client unsure of these aspects of the purpose and scope, can quickly derail a waterfall development that has moved past the anaylsis stage.
- A client may also not be happy with the delivered software product. As all the deliverables are based on the SRS and the analysis, the client may not see what will be delivered until it is almost finished.

Factors influencing the effectiveness of the agile model

Just as with the waterfall model, there are a number of factors that can occur in an agile development that can influence the effectiveness of the project.

- As discussed previously, the client in an agile development has frequent and early opportunities to see the work being delivered and to make decisions and changes throughout the project. They are required to work closely with the project team throughout the project. This high degree of client involvement, while an advantage, may present problems for some clients who do not have the time or interest for this type of participation. A client that does not have the commitment for this development model, will curtail its effectiveness.
- With the intensity involved in an agile development (especially when it is implemented using a method such as Scrum), team members need to be focused and dedicated to their work on the project. While it is not necessary that all of the tasks need to be completed in a particular sprint, if the completion of tasks drags on, and additional sprints need to be added to the allotted project timeline, this can add to the overall cost of the solution. It may also put pressure on developers that cause the quality of the solution to decline.
- A close working relationship within a software development team using the agile model is vital. If circumstances caused members of the team to work remotely or to miss work for a period of time, the effectiveness of the process will be put at risk. Webcams or online collaboration tools will alleviate this to some extent, but the daily scrum and the ability to actively problem solve or advise others will be limited.

Factors influencing the effectiveness of the spiral model

The spiral model is a complex one that combines aspects of both the waterfall and agile models. While its primary focus is on mitigating risk, its complexity leaves it susceptible to a number of factors that can limit its effectiveness.

- The complexity of a spiral development places a significant pressure on the software developer managing the project. If the person doing so is not able to navigate and guide their team through the various iterations of the development or is not able to evaluate and assess the risk at

each point, the development could be compromised.

- Ironically, as the spiral model is focused on risk management, risks that arise during the development may severely affect the project and in extreme cases, may cause the project to be ended. In many ways, it could be argued that the spiral model is doing its job if such an eventuality occurs, but if either of the agile or waterfall models were being used instead, the development may have continued on to a positive outcome.

Context Questions

1. Why is it beneficial to have an awareness of the evaluation criteria for a software solution prior to its completion?
2. In what ways can data be gathered to evaluate a software solution?
3. Why is testing such an important step in developing a software solution?
4. What is the difference between alpha testing and beta testing?
5. What test data should be selected when designing a testing table?
6. Under what circumstances is testing with live data most beneficial?
7. Of all of the factors that can have an effect on a project plan, which do you think would have the greatest potential impact and why?
8. What is specification creep and in what ways could it be prevented?

Applying the Concepts

- Compile a list of infamous bugs or logic errors that have occurred and what the consequences were in each case.

Key Skills Checklist

At the conclusion of this chapter, you should be able to address the following key skills. Mark each off as you can achieve them.

List and describe efficiency measures of a software solution ☐

List and describe effectiveness measures of a software solution ☐

Describe methods that can be used to test the usability of solutions ☐

Describe ways in which testing can be documented and what actions should take place in each case ☐

List factors that influence the effectiveness of development models ☐

Describe ways in which the progress of projects can be recorded and plans amended ☐

Discuss strategies that can be used to evaluate software solutions and project plans ☐

Sample Examination Questions

The following sample examination questions can be attempted to test your knowledge of the content of this chapter.

Question 1
What tool can be used to test to see if an algorithm is producing the correct set of outputs for a set of inputs?

- **A.** Storyboard
- **B.** Mock up
- **C.** IPO chart
- **D.** Trace table

Question 2
What does the process known as benchmarking achieve?

- **A.** Tasks within the organisation are compared to best practice
- **B.** Results of tests within the organisation are published online
- **C.** Processes within the organisation are timed and compared to their main competitor
- **D.** The main bench within the organisation is 'marked up' to show when tests were carried out

Question 3
A software solution is struggling to be effective with the main problem being the timeliness of data. This means:

- **A.** Data is too large to be processed
- **B.** Once it has been processed, the timeframe of its usefulness has passed
- **C.** The data is not being communicated clearly
- **D.** The data does not have data or time fields attached to it

Question 4
When using test data to test the boundaries of the age input for a student between 12 and 18 years old, which combination of test data would do this appropriately?

- **A.** 12, 13, 17, 18
- **B.** 12, 18
- **C.** 11, 12, 18, 19
- **D.** 11, 12, 13, 17, 18, 19

Question 5
An App is being developed that will allow a landscape gardener to take measurements of a garden and then develop a quote for the customer in real time. Describe the scope of the proposed software solution by listing **two** measures of efficiency and **two** measures of effectiveness.

Efficiency	Measure 1: Measure 2:
Effectiveness	Measure 1: Measure 2:

4 marks

Question 6
The beta version of a new software solution has just been completed by Killabrew Solutions for their client #NoLimits Real Estate. Ameer has been managing the project and is now trying to determine what criteria should be used for the acceptance testing with the client. List two criteria that must be included.

Criteria 1: _____

Criteria 2: _____

2 marks

Sample Examination Answers

Question 1
Answer: D

Question 2
Answer: A

Question 3
Answer: B

Question 4
Answer: C

Question 5
An App is being developed that will allow a landscape gardener to take measurements of a garden and then develop a quote for the customer in real time.

Describe the scope of the proposed software solution by listing **two** measures of efficiency and **two** measures of effectiveness.

Efficiency	Measure 1: It is easy for the landscape gardener to enter the values into the App. Measure 2: The quote can be easily delivered electronically to the customer.
Effectiveness	Measure 1: Quoted is calculated in real time for the customer. Measure 2: Quote is correct (comparing it to previous methods of producing quotes).

Question 6
a. The software performs the tasks required of it.
b. The software is easy to use.

Questions like this provide very little information and it is important to remember that simple answers are often the best. The question also uses says 'list two criteria that MUST' – which means that the focus is on the most important things as opposed to things that are down the list.

Chapter 9
The law in a software development context

The chapter covers Unit 4: Area of Study 2 key knowledge:

- *Interactions and impact*
    - *KK4.7 Reasons why individuals and organisations develop software, including meeting the goals and objectives of the organisation*
    - *KK4.8 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*
    - *KK4.9 Ethical issues arising during the software development process and the use of a software solution*

Key terms: Privacy Act 1988, The Privacy and Data Protection Act 2014, Health Records Act 2001, Copyrght Act 1968, cloud storage, open source software, creative commons

Study roadmap

	Unit 3		Unit 4	
Ch	Area of study 1 – Programming	Area of study 2 – Analysis and Design	Area of study 1 – Development and Evaluation	Area of study 2 - Cybersecurity: software security
1	✓			
2	✓			
3	✓			
4		✓		
5		✓		
6		✓		
7			✓	
8			✓	
9				✓
10				

Legal issues

There are a number of laws that have direct bearing on software developers. The laws that are listed in the VCE Computing Study Design are the: Privacy Act 1988, the Privacy and Data Protection Act 2014, the Copyright Act 1968 and the Health Records Act 2001. While it is important for students to have an awareness of the contents of each of these laws or acts and be able to discuss their implication on software developers, it is not necessary to memorise the specific detail or clauses within them.

The challenges faced by the legal community in keeping up with computer-related crime and privacy have been huge. The rapidly changing landscape, especially with the advent of the Internet, has meant that laws that have served the Australian community well for many years have suddenly been rendered almost

Figure 102: Australian High Court

irrelevant. The difficulties encountered because of the borderless nature of the Internet have been many; extending from the powerlessness of our Australian laws to counter outside threats to the daunting task of law enforcement.

Let's examine each of the relevant laws in turn.

The Privacy Act

Incorporating:
The Privacy Act 1988,
Privacy and Data Protection Act 2014

The Privacy Act 1988 (incorporating the Privacy and Data Protection Act 2014) is a law that sets out the ways in which organisations can collect, use or distribute personal data. The Act also outlines what must be done to protect the data and the rights of individuals with respect to their own data.

The Privacy Act applies to government organisations, local councils or any organisation that is contracted to a government organisation. The Privacy Act also applies to the private sector (more on this later).

The Privacy Act 1988 initially set out 11 Information Privacy Principles (IPPs) and 10 National Privacy Principles (NPPs). The IPPs applied to the government sector and the NPPs to the private sector. The Privacy Amendment (Enhancing Privacy Protection) Act 2012 and the Privacy and Data Protection Act 2014 combined these two sets of principles into 13 Australian Privacy Principles (APPs) which serve as the base line for the privacy legislation. Each is described in the following section.

APP 1: Open and transparent management of personal information

Data needs to be managed in an open and transparent manner. This includes having an easy to follow, accessible (free of charge and easily available) and current privacy policy. An organisation must have a privacy policy that contains information on the kinds of personal information it collects, how an individual may complain about a breach of the privacy policy and whether the organisation is likely to disclose information to overseas recipients.

APP 2: Anonymity and pseudonymity

In making enquiries or complaints, individuals have the option of not identifying themselves, or of using a pseudonym. There will of course be times when this is not appropriate or possible.

APP 3: Collection of solicited personal information

An organisation must not collect personal information (other than sensitive information) unless the information is required for one or more of the organisation's functions or activities. Sensitive information must only be collected with an individual's consent. The collection of sensitive information must also be required for the organisation's functions or activities.

APP 4: Dealing with unsolicited personal information

This may seem like an odd inclusion in the Australian Privacy Principles, but APP 4 outlines what organisations must do with unsolicited personal information (that is, personal information that they receive that was not asked for). Where an organisation receives unsolicited personal information, it must first determine if it would have been permitted to collect the information under APP 3. If so, then APP 5-13 apply as they would normally.

If the information could not have been collected under APP 3, the organisation must destroy or de-identify that information as soon as they can (as long as this is both lawful and reasonable).

APP 5: Notification of the collection of personal information

APP 5 is similar to APP 1, but differs in its application. APP 5 states that the organisation must make an individual aware, at the time or as soon as possible after, they collect their personal information.

They are also required to notify individuals about the access, correction and complaints processes in their privacy policies, as well as the location of any overseas recipients of individuals' information.

APP 6: Use and disclosure of personal information

As was the case with the previous Privacy Act, an organisation must only use or disclose information for the purpose for which it was originally collected. The new privacy legislation introduces a small number of exceptions when an organisation can use or disclose personal information for a different purpose. These exceptions include to assist in locating a missing person, to defend or establish a legal claim or for the purposes of resolving a confidential dispute.

APP 7: Direct marketing

For an organisation to use personal information for direct marketing purposes, the individual must have either given their consent or they would have a reasonable expectation that their information was going to be used for this purpose. Organisations must always allow individuals to opt-out if they wish.

APP 8: Cross-border disclosures

Before an organisation decides to disclose personal information to an overseas recipient, the organisation must take reasonable steps to ensure that the overseas recipient does not breach the APPs (excluding APP 1) in relation to that information. If the overseas recipient breaches the APPs in any way, the organisation could be held accountable as if they themselves had breached the privacy laws!

APP 9: Adoption, use or disclosure of government related identifiers

Organisations are prohibited from adopting, using or disclosing a government related identifier (think of identifiers such as Medicare numbers and Tax File Numbers).

APP 10: Quality of personal information

An organisation must take reasonable steps to ensure the personal information it collects is accurate, up-to-date and complete.

APP 11: Security of personal information

An organisation must take reasonable steps to protect the personal information it holds from interference, loss, damage, unauthorised access, modification and disclosure. This also includes destroying or de-identifying personal information that the organisation no longer needs (unless it is required to keep it for legal reasons).

APP 12: Access to personal information

An organisation must give an individual access to the personal information that it holds about that individual, unless an exception applies. In addition to giving an individual access, an organisation must respond to this request in a reasonable amount of time. If an organisation decides not to give an individual access, it must provide reasons why this is the case and make available means for the individual to complain if they wish to do so. An organisation charging a fee to give an individual access to their personal information, must ensure that the charge is not excessive. The charge must also not be as a consequence of simply making the request.

APP 13: Correction of personal information

Organisations must take reasonable steps to ensure that personal information is accurate, up-to-date, complete and relevant. This could be initiated by the organisation or could be in response to a request from an individual. In

either case, it is then the organisation's responsibility to ensure that these changes are also provided to outside entities that they share data with.

The Privacy Act 1988 was originally written for the government sector and is mandatory for government organisations and for organisations that are working under a government contract. In relation to the private sector, the Privacy Act is mandatory for the following organisations:

- Organisations working under a government contract.
- Organisations with a turnover of more than three million a year (including not for profit organisations)*.
- Organisations of any size that store medical information.
- Organisations of any size that distribute or sell personal information.

* Note that the great majority of businesses in Australia do not have a turnover of greater than 3 million dollars per year. Businesses that fall into this category are not mandated to follow the Privacy Act (unless of course they are store medical information or sell information to other companies). Organisations can choose to opt-in if they wish.

Figure 103: Medical data

Health Records Act 2001

The Health Records Act 2001 is an additional Act to the Privacy Act (although not a part of it) that gives individuals protection of their own medical information regardless of the size or type of the organisation that is collecting this data. The creation of the Act was seen as necessary as health records are considered amongst the most sensitive of all personal data. There also needed to be uniformity across both the public and private sectors. The Act establishes 11 Health Privacy Principles (HPPs) that are similar in style to the APPs. The important aspect of the Act to emphasise is that any organisation in Victoria that deals with medical information is bound by the Act.

There are very few exemptions under the Act. Even organisations such as gymnasiums, child-care centres, educational institutions, counsellors, insurers and superannuation providers are bound to follow the Act as they all collect personal health information of one kind or another.

The legislation is not all in favour of the individual. Under the Health Records Act, organisations collecting medical information can share information with other organisations for the purposes of research and planning as long as the information is de-identified. In addition, the Act allows for the mandatory reporting of certain diseases without an individual's consent (such as STDs and specific infectious diseases).

Copyright Act 1968

In Australia, copyright laws are governed by the **Copyright Act 1968**. In 2004 there were significant changes made to this act in Parliament, particularly as it relates to computer technology.

Copyright in Australia is free and automatic. That is, if you are the creator of a work, you do not have to place a "(c)" symbol on the work itself or register that you are the copyright owner with any organisation or body. If you are the creator of a work, you own the copyright automatically.

An owner of the copyright over a piece of work has a number of rights. They have the right to choose when and how the work will be distributed, published or otherwise communicated. They also have the right to incorporate any sort of technological protection devices to protect the work.

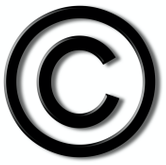


Figure 104: The accepted symbol to denote copyright

When dealing with a copyrighted work, it is illegal to copy or share the work without the permission of the author of the work. It is also illegal to change the format of the work (digitise it by scanning it into a digital form, change the file format or use a compression or ripping technology). If a copyrighted work is protected in some way, it is illegal to circumvent this protection or use a tool to do so.

Individuals do have some ways that they can use a copyrighted work without breaching the Act. They can copy 10% or 1 chapter of a reference book (whichever is greater) without permission. A copyrighted work can be used under the banner of "fair dealing" for parody, review, research, reporting or by the legal system.

The Copyright Amendment Act 2006 was an amendment to the Copyright Act brought about by the US and Australia Free Trade Agreement.
In particular:

- The recording of TV or radio programs for family or personal purposes was allowed as an exception under the Act.
- Format shifting of music for personal use was also allowed as an exception under the Act. Without this exception, ripping legally purchased CDs to an iPod would be a breach of the Act.
- Prior to this amendment, a copyrighted work entered the public domain 50 years following the death of the last author or creator of the work. As the legislation in the US is 70 years instead of 50 years, the Australian Copyright Act was changed to 70 years to bring it in line.
- The concept of "fair dealing" (which in the US is called "fair use") was introduced in this amendment.

Open Source Software (OSS) / Creative Commons (CC)

As a software developer, it is not always necessary (or desirable) to "reinvent the wheel". Fortunately, there are code libraries that are freely available on the Internet that enable software developers to quickly code routines that would otherwise be time consuming, expensive and tedious to create. This is a very different concept to simply looking on the Internet for some code and "taking it". Doing that could very well be a breach of copyright. Software that has been created (and tagged) as OSS has been created with this very purpose in mind, but there are some catches.

OSS is often protected by a licence of some type and a number of these exist. The organisation known as Creative Commons has created a number of licences that owners of any sort of work can attribute to the work to waive some or all of their copyright rights. These licences have been translated for many countries and a set of these licences exist for Australians to use specific to Australian copyright law.

The following licences exist:

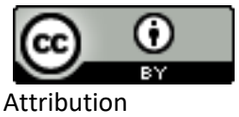


Figure 105: Attribution

This licence allows others to use, modify, distribute (including commercially), as long as the original author is credited. All of the content for the description of these licences and the icons themselves are under an attribution licence – and have been sourced from the Creative Commons website.

Figure 106: Attribution, Share Alike

Once again, this licence allows others to use, modify (including commercially) and distribute the work as long as the original author is credited. In addition to this, the new product must be shared under the same terms. This is the licence that is closest to a true OSS model.

Figure 107: Attribution, No Derivatives

This licence allows others to use and distribute (including commercially) as long as the original author is credited. However, no derivative works are allowed. That is, the work must be unchanged.

Figure 108: Attribution, Non-Commercial

This licence allows others to use, modify and distribute the work; however it cannot be distributed commercially. The author must of course be credited, but interestingly, derivative works do not need to use this licence.

Figure 109: Attribution, Non-Commercial, Share Alike

This licence allows others to use, modify and distribute the work non-commercially as long as the original author is credited. Any derivative works must also be shared in this way with the same licence.

Figure 110: Attribution, Non-Commercial, No Derivatives

This is the most restrictive of the licences as it allows others to use and distribute the work non-commercially but not modify the work in any way. The author must be credited.

The legal implications of cloud storage

The use of cloud storage has many benefits for organisations, the least of which are portability of data and the security of off-site backup. There are however legal issues that must also be taken into account. In response to the Privacy Act, organisations must ensure that they take reasonable steps to keep the data secure (APP 11). If the data is of a medical nature, the adherence to this APP is not mandatory.

The difficulty with the use of cloud services is that an organisation is effectively handing over the control of their data to an external vendor. Personal data of any kind has a "street value" to hackers as it can be used to perform identity theft.

Small to medium sized businesses that are not able to invest a great deal of capital into security infrastructure, may find cloud services attractive and affordable. Doing so may increase the level of security over what they would be able to provide. Under the Privacy Act, this would seem to be a reasonable.

As many cloud services are not based in Australia, an organisation planning on storing personal data in this way needs to ensure that

the organisation adheres to the Australian Privacy Principles (per APP 8). The use of an Australian cloud provider will certainly make the process simpler, as they will already be aware of the Privacy Principles. In the case of cloud servers located in other countries, different Privacy Laws will be in effect. In this case, organisations are responsible for ensuring that the provider is adhering to all of the applicable APPs in the Australian Privacy Act. This will usually take the form of a contract in which the organisation will define their requirements in terms of how the data is handled and accessed. If an organisation has taken "reasonable steps" to ensure that the cloud storage provider is keeping the data secure in accordance with Australian law, then they will not be liable for any data breach.

An organisation would also be wise to do the following:

- Ensure that the cloud provider is a reputable vendor (some cloud providers abide by the International Standard for Cloud Privacy – ISO270018)
- Encrypt data sent to the cloud
- Use multi-factor authentication to access the data
- Maintain an encrypted backup with a different cloud provider
- Ensure that the ownership of the data remains with the organisation

Why do organisations or individuals develop software solutions?

In Chapter 4, we discussed at length how the mission statements of organisation inform their goals and objectives. In putting these goals and objectives into action, organisations will create information systems that will in turn, have their own set of goals and objectives.

Individuals will create software solutions for similar reasons. They will have a problem they wish to create a solution for, and this solution will be in service of some goal or objective that they have.

Software development is generally driven by specific need, whether this be the need of individuals or organisations. The reasons why software solutions are developed probably fall into one or more of the following categories.

- There is a need in the market and doing so will be profitable.
- To remain competitive.
- It will be beneficial to society.
- To perform a task that is difficult or not possible otherwise.
- To assist with the transition between information systems.
- To save time.
- To save money.

The motivations of individuals or organisations involved in developing software will sometimes put them at odds with the law or other stakeholders (both internal and external). It is also possible for individuals or organisations to conduct themselves in ways that are legally sound, but still considered to be unethical.

As a software developer, ensuring that the law is followed, specifically in relation to the software solution that you are creating, is very important. This can, however, bring a software developer into conflict with other stakeholders involved in the project – either though a misunderstanding of a software developer's intentions or by insisting on the inclusion (or exclusion) of features that would contravene the law. There are also times when a situation may challenge a software developer in an ethical way and there is often quite a difference between what is unethical and what is illegal.

Some examples of this sort of conflict follow, although this list is by no means exhaustive.

Ethical issues arising in the software development process

The Privacy Act requires that a reasonable level of security be placed on personal information that is being stored by a software solution or in a networked environment. For this reason, it is often important to include password protection, levels of access and encryption of some sort. Organisations often have a password policy, which they may require a software developer to enforce in a software solution. Typical password policies now consist of upper and lower case characters, numbers and punctuation and are often a set minimum length. These password policies often cause concern amongst users as they are hard to remember and seem to be "over the top". A software developer may encounter resistance from staff when implementing a policy such as this or may encounter problems if their software solution uses a different policy or none at all.

What if a software developer were asked not to implement security features in a program due to time or budget constraints?

What if a software developer, in the course of the creation of the software solution, found that the security of information on the networked environment was inadequate?

What if a software developer were asked by the client to include features within the program that could potentially breach the privacy of the user – such as key logging software or the use of an in-built camera or microphone without the user's knowledge?

A software developer may be asked to forego rigorous testing in favour of a reduced timeline or a reduction in costs. It may even be tempting for a software developer to not thoroughly test a software solution as this part of the software development process is often not documented for the client.

Documentation for the client catering for different groups within the organisation is very important but may be a part of the project that either the client or the software developer wants to omit.

Code of ethics

Most professions are also guided by a code of ethics that lays out guidelines specific to the profession. A code of ethics is not legally binding but sets out to establish a level of ethical conduct and to raise the awareness of issues related to the profession.

The Australian Computer Society, which represents a large proportion of the IT professionals in Australia, has a published code of ethics can that can viewed via their website.

The introduction of the ACS code of ethics states "An essential characteristic of a profession is the need for its members to abide by a Code of Ethics. The Society requires its members to subscribe to a set of values and ideals which uphold and advance the honour, dignity and effectiveness of the profession of information technology." Within the code of ethics are statements about honesty, competence, values, ideals, professional conduct and social implications.

Context Questions

1. Do all private companies or organisations need to follow the Privacy Act?
2. Under what circumstances is a private company exempt from adhering to the Privacy Act?
3. What changes were made with the introductions to the amendments to the Privacy Act 1988?
4. If an organisation shares your personal information with a company overseas (with your permission) and the company breaches one of the APPs, is the original organisation liable? Explain.
5. What rights does the author of a work have?
6. Under what circumstances can medical providers share your personal information without your permission?
7. How much of a copyrighted work can be copied without the permission of the author?
8. What is the relationship between Open Source Software and Creative Commons?

Applying the Concepts

- Find examples of OSS or creative commons content and note the conditions under which the authors have said it can be used.

Key Skills Checklist

At the conclusion of this chapter, you should be able to address the following key skills. Mark each off as you can achieve them.

List and describe efficiency measures of a software solution ☐

List and describe effectiveness measures of a software solution ☐

Describe methods that can be used to test the usability of solutions ☐

Describe ways in which testing can be documented and what actions should take place in each case ☐

List factors that influence the effectiveness of development models ☐

Describe ways in which the progress of projects can be recorded and plans amended ☐

Discuss strategies that can be used to evaluate software solutions and project plans ☐

Sample Examination Questions

The following sample examination questions can be attempted to test your knowledge of the content of this chapter.

"Even though I created this module for my last job, it's mine to sell and do what I wish with".

Question 1
This person would be breaking the law if they were to do this. Which law would this person be infringing?

- **A.** Privacy Act
- **B.** Code of Ethics
- **C.** Copyright Act
- **D.** Health Records Act

Question 2
The winners of a tennis tournament are to have their names and photos published on the club website. This is not a breach of privacy legislation as long as:

- **A.** Other tennis clubs are doing it as well
- **B.** The players are members of the club
- **C.** The players have a social media account
- **D.** The players have given their permission for this to happen

Question 3
Open Office, VLC and Audacity are examples of Open Source software packages. This means:

- **A.** The software and source code is free to use and access
- **B.** A small licence fee is required to use the software
- **C.** The software can only be used by developers
- **D.** The software has to be programmed for it to be used

Question 4
A new App asks users to create an account and supply an address for security verification. This is in breach of the Privacy Act because:

- **A.** it is a unique identifier
- **B.** another user may have supplied the same address
- **C.** the owners of the App would be unable to check the accuracy of the address
- **D.** the owners of the App have no valid reason to collect this information

Question 5
What rights does an author of a work **not** have?

- **A.** The right to chose when and how the work will be distributed
- **B.** The right to incorporate protection methods to protect their work
- **C.** The right to control how their work is communicated to others
- **D.** The right to prevent anyone quoting from the work

Question 6
Paul is in the process of developing an App that will be free to download. One of the ways that the App will revenue is via advertising. Paul wants to make the advertising as targeted as possible by accessing the browsing history of those using the App, as well as their location details. Paul's colleague Maxine advises Paul this this is technically very easy to code, but that some things need to be done in order to 'make it legal'.

a. What would be the best way to ensure that this data is able to be obtained legally?

_____

_____

1 mark

b. List two potential negatives that could arise from implementing such a feature.

Negative 1: _____

_____

Negative 2: _____

_____

2 marks

Question 7
AFLellofAGame is an AFL based podcast that is recorded weekly. The show is broadcast live and then a recording is placed on a server for access. Both the live show and archived shows can be accessed via an App that can be downloaded free from the App store. The hosts have taken to starting the show with a pop song to represent what has happened during the week. Two weeks ago, they played most of "Take it to the limit" by The Eagles. Last week the song they selected was "Firework" by Katy Perry. Paul (one of the hosts) was initially concerned that this might be a breach of copyright, but Dave (the other host) reassured him saying that by stopping the song just before the end meant they weren't playing the whole song. He also said that as they weren't selling the song, that it was OK.

Discuss any problems with Dave's reasoning, with reference to any relevant legislation.

_____

_____

_____

2 marks

Question 8
The web developer for a company is approached by the another company with a proposal. They would like to have their company advertised on the website and in exchange will share their client data. The web developer feels that the proposal has merit, but also knows that some there are some things to be careful of legally.

a. What piece of legislation is most relevant to this issue?

_____

1 mark

b. Describe what the web developer needs to do in order to ensure that this proposal is implemented legally.

_____

_____

2 marks

Sample Examination Answers

Question 1
Answer: C

Question 2
Answer: D

Question 3
Answer: A

Question 4
Answer: D

Question 5
Answer: D

Question 6

a. The best way to obtain this data legally is to advise listeners downloading the App that this data will be collected and ask for them to agree to this.

b. Negative 1: the App might not be as responsive as it will be sending additional data back and will also be loading ads to display.
Negative 2: listeners might get annoyed at the presence of the ads and may also feel that their privacy is being infringed upon.

Question 7
It is against the Copyright legislation to play the songs that Dave is playing at the front of his show. Stopping the song just before the end makes no difference. The station may not be selling the song, but they are making revenue from advertising used in the show, so the song is contributing to that (so indirectly – they in fact are selling the song).

Question 8
a. Privacy legislation.
b. Clients need to be asked whether they will allow their data to be shared with and informed as to what it will be used for. If they say no, then a record of this must be kept and the data cannot be shared.

Chapter 10
Cybersecurity

The chapter covers Unit 4: Area of Study 2 key knowledge:

- *Digital systems*
    - *KK4.1 Physical and software controls used to protect software development practices and to protect software and data, including version control, user authentication, encryption and software updates*
    - *KK4.2 Software auditing and testing strategies to identify and minimise potential risks*
    - *KK4.3 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*
    - *KK4.4 Types of web application risks, including cross-site scripting and SQL injections*
    - *KK4.5 Managing risks posed by software acquired from third parties*
- *Data and information*
    - *KK4.6 Characteristics of data that has integrity, including accuracy, authenticity, correctness, reasonableness, relevance and timeliness*
    - *KK4.10 Criteria for evaluating the effectiveness of software development security strategies*
    - *KK4.11 The impact of ineffective security strategies on data integrity*
    - *K4.12 Risk management strategies to minimise security vulnerabilities to software development practices*

Key terms: security threats, accidental, event-based and deliberate threats, risk management, user authentication, dictionary attack, rainbow table, firewalls, malware, SQL injection, man in the middle attack, social engineering, encryption, logical and physical security, barrier techniques, biometrics.

Study roadmap

	Unit 3		Unit 4	
Ch	Area of study 1 – Programming	Area of study 2 – Analysis and Design	Area of study 1 – Development and Evaluation	Area of study 2 - Cybersecurity: software security
1	✓			
2	✓			
3	✓			
4		✓		
5		✓		
6		✓		
7			✓	
8			✓	
9				✓
10				✓

The nature of threats

When considering the nature of threats to an information system, they can be placed into three broad categories: accidental, deliberate and event-based. Let's have a brief look at accidental and event-based threats first, as these are the simplest ones to describe.

Accidental threats

Data can be lost in many different ways. Of all the threats to data that will be discussed in this

chapter, the most common way that data is lost is through the deletion of files or parts of them without having any backups available. Updating and deleting files is an activity that people do almost every day and so it is easy to make a mistake or not think twice about it.

> *Security threats: The actions, devices and events that threaten the integrity and security of data and information stored within, and communicated between, information systems. The threats can be accidental, such as losing a portable storage device containing files; deliberate, such as malware, phishing; and events-based such as a power surge.*
>
> *VCAA VCE IT Study Design 2020-2023 Glossary*

Files can be saved in the wrong files formats, causing others to think they are corrupt. It could also be administrative errors or processes that cause files to be lost or corrupt.

Data can also be lost accidentally through the physical loss of hardware. Notebooks, smart-phones and flash drives are all easily lost or misplaced and sometimes this can mean that sensitive data is exposed to risk.

Event-based threats

Event-based threats are ones that are categorised by particular events occurring that are outside of the control of users. These could be on a small scale such as the failure of a hard drive, a power failure or software freezing causes a file to become corrupt (or not able to be saved). With the wide-spread use of cloud services, a business providing cloud storage or software as a service could experience a loss of their own or simply go out of business.

Event based threats can also be on a much larger scale. Natural disasters such as fire, earthquake, flood or tornado, could all lead to quite significant losses of data.

The loss from accidental damage as well as event-based damage are best prevented with good backup and operational procedures (as discussed in Chapter 8). However, deliberate threats require the use of security measures to minimise their risk.

Deliberate threats and security

What is security? No matter what your definition and how complex it may be, security is ultimately about prevention. Computer security seeks to prevent the damage that could be caused if unauthorised people were to gain access to an organisation's information (known as a deliberate threat).

When discussing the topic of security and the nature of deliberate threats, there is not one absolute way to protect the integrity of data and information. This is due mainly to the fact that there are so many different types of deliberate threats. Deliberate threats are present from both inside and outside of an organisation. They can be both physical and logical in nature. There are ways to protect against certain types of threats, but a 'solution' to this problem is complex and one that needs to be constantly attended to – otherwise the threats can advance beyond the capability of the defences.

When discussing this topic, it is natural to think of the threats from the 'outside' (or the Internet) being of the most concern. While these threats are definitely a concern, it is more common to encounter threats from within an organisation, but more on this later. Let's start by looking at the threats from the Internet.

Why do Internet threats exist?

The Internet was built for a trusted world. The beginnings of the Internet (ARPANET) were established by the US military using packet switching to enable a network to operate if nodes within the network went offline at any time. The technology was essentially built to be 'nuclear-war' proof – so that in the event of a nuclear strike with key US military installations

being destroyed, the network would remain in operation and the US would be able to launch a counter-strike. For a time, the network was made available for select universities to use and was opened up to commercial interests in the late 1980s. In the early 1990s CERN publicised the 'world wide web' project and the Internet began to gain in popularity and make its way into the world culture.

The underlying engineering behind the Internet was not built with security concerns in mind. There was no e-commerce, no crime and no financial transactions. HTTP is not a secure protocol and contains no encryption of any kind. Internet security, as a result, had not been a coordinated effort.

The ease with which Internet sites can be cloned makes trust a difficult issue. It is not enough to trust a web-site based on the reputation of the 'brand'. Prime examples are banking sites, which are regularly cloned using very similar URLs and by using the images captured from the official web-site.

Internet security risk management

Internet security, for many organisations, falls into the area known as risk management. That is defined as the application of security measures based on a cost/benefit analysis. For example, there may be many security measures that can be applied to a company's network, but at some point in time, the cost of adding the security measure will be greater than the potential losses that can result. In many ways, risk management is about probability. Ultimately, no network or organisation is 100 per cent secure and it can cost an organisation a very large amount of money if attempting to attain this ideal. At some point, an organisation needs to decide that what it has in place is 'good enough' and that they have sufficient data recovery methods in place to ensure that they can recover from a worst case scenario.

With any security measure that is applied to a system, there is a performance cost as well as a monetary cost. This also needs to be factored into the cost/benefit analysis.

Protection and detection

Security is primarily concerned with protection but it also includes detection. Protection is what we naturally assume when we raise the topic of security; that is, what can be done to prevent malicious attacks on the data within an organisation. Protection is often at the 'front door' – the problem being that once an intruder has access, there are no mechanisms to detect their presence. This is where detection is important.

One of the first barriers that can be effective in protecting an organisation and its data, is user authentication.

User Authentication

User authentication is the process by which a person that is wishing to gain access to an information system provides satisfactory credentials to allow them to be confirmed as being who they say they are. User authentication falls into three categories: ownership factors (based on something the user has), knowledge factors (based on something the user knows) and inherence factors (based on something the user is).

Some examples of these are as follows:

Ownership factors: ID card, security token, mobile phone, implanted device.

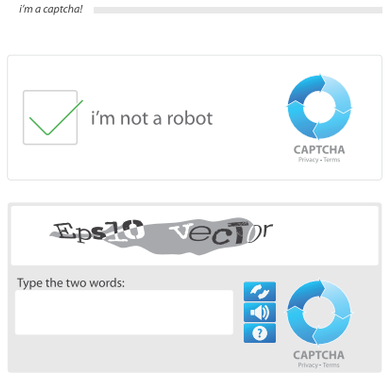


Figure 111: Example of a captcha.

Knowledge factors: a password, a pass phrase, a Personal Identification Number (PIN), answer to a security question, response to a question (such as a 'captcha').
Inherence factors: fingerprint, retinal pattern, DNA sequence, signature, face, voice.

While each of these categories of user authentication have their strengths and weaknesses, many organisations are moving to the use of multi-factor user authentication. This involves the use of 2 or more forms of authentication, ideally from different categories. For example, a common (and easy to implement) multi factor authentication is to ask the user to enter their username and password, and follow this with a pass code sent to their mobile phone.

Password construction

Many organisations enact password policies that force users to create complex passwords that follow a number of rules. Typically this involves using at least 8 characters, a mix of numbers, alpha-numeric characters and special characters as well as upper and lower case characters. Policies vary, but passwords of this sort are considered 'strong' by organisations and they will often have an interface that will give a user feedback on the relative strength of their selected password against this measure. The formation of such policies is done by considering the 'entropy' of passwords constructed using the set of rules laid out.

As the entropy of each character is multiplied by the number of characters, password length is one of the most important factors in creating a secure password. There is a fine balance, as passwords can quickly become too long, complex and hard to remember. Best practice for password construction generally involves using something memorable to the user but not easily guessed by anyone else.

> *Password entropy is a measure of how easy a password will be to 'crack'. A known password will have an entropy of 0 bits. One that can be guessed 50% of the time on the first guess will have an entropy of 1 bit. A password's entropy can be calculated by finding the entropy per character (which is log base 2 of the number of characters in the character set), multiplied by the number of characters in the password.*

Cracking passwords

One of the simplest ways to 'crack' a password is by using 'brute force'. Brute force is a method by which every combination is tested. While this would be an arduous task for a human being to perform, it is very easy to write code to try out all of the combinations possible. The level of difficulty rises with the different combinations that are allowed in the

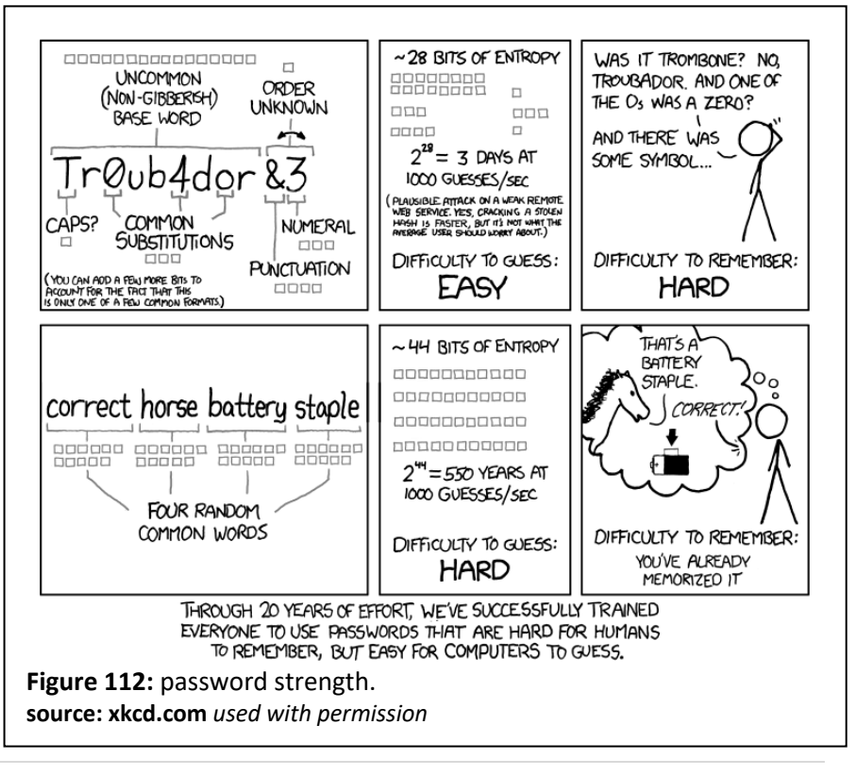


Figure 112: password strength.
source: xkcd.com *used with permission*

password. This is the reason why 'complex' password policies add addition character sets (such as upper and lower case, numbers and special characters). However, we have already discussed how adding these characters doesn't increase the entropy per bit by very much. It is really the length of the password that makes 'brute forcing' a password difficult (especially if the length is unknown).

Many systems will include a mechanism that will suspend an account after a small number of incorrect attempts have been made, and this is usual enough to make 'brute forcing' ineffective.

Dictionary attack

A more common method of attempting to crack a password is a dictionary attack. A dictionary attack is based on trying all the strings in a pre-arranged list. Dictionary attacks often succeed because people have a tendency to choose short passwords that are ordinary or common words, or simple variations that include a digit or punctuation character on the end. Dictionary attacks can easily be thwarted by choosing a password that is not a simple variant of a word found in any dictionary or listing of commonly used passwords.

Storing passwords securely

While we have been discussing methods by which users can create passwords to protect themselves, the best password in the world will be of no protection if the password database and where it is stored in the information system are not secure.

In the early days of software development, usernames and passwords were stored in databases and text files within the information system. While these files were often secured in some way, administrator accounts could be targeted for brute force or dictionary attacks in order to gain access. Instead of targeting individual user accounts, by targeting an administrator account, a hacker could gain access to all of the usernames and passwords stored in the system. A solution to this is hashing.

Encrypting data using hashing

We will discuss how encryption works shortly, but for the purposes of storing passwords securely, hashing is the best method. Hash tables were discussed in Chapter 3, and the method for hashing passwords is the same.

Hashing works in the following way. When a user enters their username and password, both are hashed using a key to produce a hashed value of each that is significantly different from the original. This hashed value can be compared to the hashed value that is already stored in the password database. The main benefit of using this method is that the database only contains the hashed values. As the hashing process is a one-way conversion, gaining access to the hashed password database is not very useful if the hash key is not known. Of course, once the database is obtained by a hacker, they can take their time and brute force different hash table keys, so while this is far better than storing usernames and password in plain text, it is still not very secure. Once a hashed database is obtained, rather than brute force, a hacker will make use of a rainbow table.

Rainbow tables

A rainbow table is a listing of all possible permutations of encrypted passwords specific to a given hash algorithm. Once a hacker gains access to an information system's password database, they compare the rainbow table's precompiled list of potential hashes to the hashed passwords that are stored in the database. The rainbow table is effectively a reverse engineering of the hash that has been applied to each username and password. It is an offline only attack, so while much more powerful than brute force or a dictionary attack, a hacker must first have been able to gain access to the password database.

Rainbow tables can be used to crack 14-character alphanumeric passwords in about

160 seconds. Unfortunately for those designing the security of information systems, the increase in processor speed and data access times have meant that large rainbow tables are easily viable. Large rainbow tables can approach 700 GB in size and vary depending on the hash algorithm that is being targeted. While the hacker will have to guess which has been used, they may simply try many rainbow tables.

To protect against the use of rainbow tables, system administrators use 'salt'.

Salt

'Salt' is the process of adding random data to each username and password before it is hashed. This has the effect of making passwords and usernames longer (which adds to their complexity) which results in a very large hashed value. Rainbow tables have great difficulty with salted hashes as they need to not only be able to work out what was used, but also what salt has been added.

The best 'salts' are uniquely random ones that are applied to each username or password (as opposed to the same 'salt' being used for everything).

Version control and software updates

By maintaining a regular system of version control and ensuring that key software packages are updated where possible, the possibility of vulnerabilities existing and being exploited is minimised.

CASE tools

Keeping track of all of the software packages within an organisation can be a difficult task. A category of software tool known as CASE tools can be used by organisations to do this in an efficient manner. While the breadth of tools available under the banner of CASE tools covers all aspects of the PSM, they do have a particular application in keeping track of the versions of software packages that are being used and updates that are available. In some cases they may be automated to push updates out to users.

While we have been discussing techniques to protect information systems and data, one of the most difficult aspects of information system security is detecting when an intruder gains (or has gained) access. Often intrusions go undetected and even when detected, it is not always easy to determine what data has been accessed.

Detecting intruders

Intrusion detection can take place if particular files or assets within a network are tracked or 'watched'. Audit logs can be generated that track all network traffic and in this way an unauthorised entry into a network can be detected. The downside of this is that it can take quite a bit of effort to 'watch' files on a network. Audit logs will track all network activity, but the task of sorting through legitimate traffic and unauthorised traffic is not easy or remotely enjoyable! In this sense, it is easy to trace the activity of an intruder after a breach has been detected, but by this stage, the damage may be done. The result of detection will hopefully be the removal of the intrusion mechanism.

A honeypot is a form of trap that is used to detect hacking attacks or collect information on malware that can be used to protect the information system against future attacks.

Honeypot intrusion detection

The 'honeypot' intrusion detection method can be used to steer intruders away from the heart of the network, gather information that will help prevent further attacks and possibly gather evidence that can be used in any potential prosecution. A 'honeypot', as the name suggests, is a server on the network that is set up as generically as possible, and often

without much protection, as a lure to an intruder to the network. It is often populated with old or useless data but in such a way that this data seems to be legitimate and valuable.

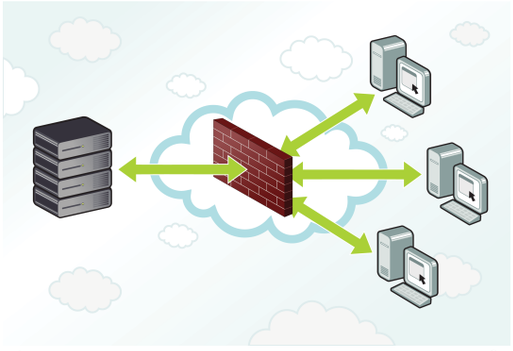


Figure 113: Diagrammatic representation of a firewall

The placement of the 'honeypot' server is often inside the firewall of the network but removed from the real servers by other barriers. The 'honeypot' will have active logs and trace software in operation and as it will not be in use by the legitimate users of the network, traffic on it can immediately alert the administrators of a network to an intrusion.

The role of firewalls

One of the first tools that organisations employ to protect themselves from outside threats is a firewall. A firewall can be software or hardware (usually as part of a router) that filters traffic based on a set of rules that can be changed to suit the environment. A firewall is good at preventing outside intrusion via ports that are not open for any legitimate traffic to or from the organisation. All traffic into or out of an organisation has to pass through a firewall. Firewalls are very good at stopping the spread of 'worms', but when it comes to other types of 'malware', they are basically ineffective.
A network firewall is installed on the boundary between two networks (usually between the Internet and the organisation's information system). A client firewall is software that runs on an end user's computer. Whereas a network firewall protects a whole organisation, a client firewall only protects the computer on which it is running. Client firewalls will often work by asking the user if a particular program that is seeking a connection into or out of their computer is to be allowed access and will quickly build a profile for that user and computer.

When traffic passes through a firewall, it is inspected to see if it meets certain criteria. If it does not meet these criteria, the traffic is blocked; otherwise it is allowed to proceed through. Firewalls can filter traffic based on the type of network traffic (known as protocol filtering), the source or destination address (known as address filtering) or the attributes of the packets being sent.

Malware

'Malware', or malicious software encompasses a wide range of software that has been designed for a purpose contrary to the wishes of the user who is 'infected'. The term 'computer virus' is often used incorrectly to describe software of this type – however a computer virus is a particular type of program and malware describes the broader category of which a computer virus is a member.
So what types of software categorise malware? Malware includes computer viruses, worms, Trojans, spyware and other types such as rootkits, but in this text, we will just discuss these main types. Some of these types include what is known as a 'payload', which is the part of the malware that damages the computer, provides access from outside or performs some other specific function.

There is a misconception that there are no viruses for the Mac or Linux platforms. While it is true that there are certainly far fewer viruses for these platforms, it is not as a result of the Linux or Mac platforms being more secure than Windows. As around 90 per cent of the world's computers are Windows-based, it makes more sense to write malware for Windows if the creator wants it to have the best chance of propagation. Linux and Mac platforms are far less attractive targets to the creators of malware who are looking to spread their software creations and ultimately steal money

or cause damage. With the popularity of Apple hardware and software continuing to rise, however, we may begin to see more and more instances of Mac related viruses.

Computer viruses

A computer virus is a program that is designed to replicate itself and is often transmitted via removable secondary storage devices such as USB thumb drives or portable hard drives. As a virus needs to be executed to perform its function (which is usually not just replication), viruses often attach themselves to legitimate programs or place themselves in locations where they will be executed (such as the boot sector of primary or secondary storage devices).

Computer viruses can operate in a number of ways. Often when the file is executed, they will seek out other files to infect before passing control to the legitimate executable file to which they are attached. Some viruses will load themselves into RAM so that any files that happen to be loaded will be infected.

Computer viruses have a number of methods by which they try to avoid detection. Some have been known to include code within them that encrypts the virus so that the 'signature' of the virus is different. Other more sophisticated viruses have been known to re-write parts of their own code for this purpose. Another way that a virus can remain undetected is if it has code that specifically tries to counter the type of scan that a virus checker does. In this case, the virus tries to intercept the scan and send an 'OK' signal back to the virus checker.

Most computer viruses exist for some purpose; that is, they have a payload. The payload, when triggered, can be as simple as deleting or corrupting files, causing a prank effect or displaying a message of some type. Viruses are often triggered some time after the infection and this trigger can be a specific date or time or can be completely random.

Worms

A worm, as opposed to a virus, does not (usually) have a purpose other than the consumption of bandwidth. A worm is a program that rapidly replicates itself through a network without the need of user assistance. Worms have been known to consume the bandwidth of organisations so much that the organisations have effectively had to shut down their network.

Some worms carry a payload and in these cases their purpose is usually related to opening a 'back door' to the computer with the controller of the worm able to take control of the infected computer and use it as a 'zombie' (see 'zombies' and 'botnets' later in this chapter).

The Conficker worm

The Conficker worm is notable as it has been one of the most widely spread infections of a worm in history. Estimates are that over 7 million computers were infected in more than 200 countries. Many of the functions of the Conficker worm are typical of worms in general, so it is a useful example to discuss.

Conficker was first detected in 2008 and it propagated via a security vulnerability in Windows XP. Conficker would not only propagate, but would also make contact with a host server to receive updated code. This was one of the aspects of Conficker that made it very difficult to eradicate. Conficker was constantly being improved via updates to counter what was being done to detect and remove it (updates were named 'Conficker.B', 'Conficker.C', etc.). The vulnerabilities that it was exploiting were added to in each version update, as were the ways in which it made contact with the update server. Technically, having an update server creates a central point connecting all of the worms and should make the eradication of the worm easier – but only if the update server can be identified and shut down. However, Conficker would seek out its update server by using a randomly generated domain name which the 'owner' of Conficker could access when they wished by registering

it and placing the update on it. As all of the Confickers were accessing random servers, the 'owner' of Conficker was only able to update a percentage of them at a time in this way, but as each update server was detected and shut down, more could be set up to continue communication between the 'owner' and the worms.

'Conficker.E' was the first to download a payload – which it did from an update server located in the Ukraine. Many believe the Ukraine to be the origin of the Conficker worm for this reason and also because the worm was programmed to not affect computers with IP addresses in the Ukraine or with Ukraine keyboard layouts!

The Conficker payload consisted of a 'scareware' anti-virus program (described later in this chapter) as well as the installation of a 'spambot' which would scan the computer for email addresses for use by spammers (called harvesting) as well as use the computer itself to send spam (which protects the spammer themselves from direct detection).

Trojans

A Trojan is a virus that is disguised as another software package that performs a different (although not always legitimate) purpose. It is named after the Trojan horse in Greek mythology, which was used to hide Greek soldiers in the Battle of Troy.

A Trojan may perform a number of different functions based on its payload. A Trojan can install a 'spambot', it can install software that allows the owner of the Trojan to access the infected computer, it may install pop-up advertisements, it can turn the computer into a 'zombie' or it can delete files in the same way a virus can. One thing that a Trojan does not do well is replicate, instead it relies on the users to download and run the program or distribute it. Despite this, makers of anti-virus software report that the majority of malware is in fact of the Trojan variety.

Figure 114: This Trojan horse sculpture was made for the 2004 movie Troy starring Brad Pitt. It was donated to the Turkish city of Canakkale where it is a tourist attraction.

Most of all of the 'cracking' programs, key generators, etc. designed to give users access to pirated software, contain Trojans of some description – as those who download these types of packages are quite trusting of the product, as it is by its nature written by an unidentified person and is usually very unpolished.

Spyware

Spyware is a type of malware that is concerned primarily with the collection of information. This information could be personal details (which could be used for identity theft), email addresses, browsing activities (for the purposes of marketing) or simple keystrokes. Spyware is often delivered as the payload via a worm or a Trojan and is not normally self-replicating. Spyware can also be installed when visiting a compromised web-site via specific vulnerabilities in Internet Explorer. In a case like this, the user could be infected and not be aware that the infection has occurred.

Adware

Adware is a subset of spyware in that it performs some of the same functions. The function that classifies adware is that it will display pop-up ads to the user and do so frequently. These pop-up ads can be targeted to the browsing habits of the user or they may

be completely unrelated. In most cases, the advertising being displayed is generating income for the owner of the adware in terms of the number of 'clicks' the advertisement is receiving.

A DDOS attack does not aim to steal or corrupt data, but instead tries to overload web servers and cause them to be unusable. This loss of service can be very costly to a company.

Zombies and botnets

One of the main purposes for malware is to establish 'zombies' which can be used to send spam (or act as a 'spambot'). This makes it harder to establish the sender of the spam – thus protecting the spammer from prosecution. It is estimated that well over half of the spam in the world originates from 'zombies'.

'Zombies' can also be used as part of a 'botnet' to carry out distributed denial of service (or DDOS) attacks. A DDOS attack is carried out to flood a targeted web-site with so much traffic that it slows down significantly or possibly crashes. 'Zombies' in a 'botnet' are all instructed to send IP (Internet protocol) requests to a targeted web server at the same time, and the sheer number of requests is designed to be in excess of the number that the server can handle. There is very little that any company can do in response to a coordinated DDOS attack of sufficient scale aside from changing an IP address – a common tactic that unfortunately is not instant, as it can take up to 72 hours to propagate a new IP. There is also no guarantee that this will prevent a DDOS attack, as the attackers can simply redirect their attack to the new IP.

As it is possible to detect when a DDOS attack is occurring and take a web-site off-line (as a prevention mechanism), 'zombies' can also perform a distributed degradation of service attack instead. In this sort of attack, the 'zombies' pulse their IP requests so that instead of flooding the web-server and inevitably shutting it down, the Internet traffic is slowed considerably. This sort of attack is much harder to detect or even do anything about.

Other types of attack

Of course, other types of attack exist where the intruder is playing a much more active role. Some examples of these follow.

SQL injection

Often in order to authenticate a user, their username and password will be compared to what is stored in a database. A query language such as SQL is used to extract the relevant data. SQL or Structured Query Language, is a database query language specifically designed to extract, add, delete or edit records in a database. The simplest command is one that extracts all of the records from a database table.

```
SELECT * FROM movies;
```

This command would extract the set of all of the database records from a table called 'movies'. The '*' character is a wildcard used to select all the fields. Instead of the wildcard, a command such as the following can be used, which lists all of the fields that you want to extract.

```
SELECT title, genre, director FROM movies;
```

When a command such as this is executed, the set of results can be iterated through or displayed, depending on what the purpose of the application.

When managing a login, an application may take the username and password that has been entered and execute an SQL query such as the one below:

```
SELECT * FROM users WHERE
        name = ' & txt_username & ' AND
        password = ' & txt_password & ';
```

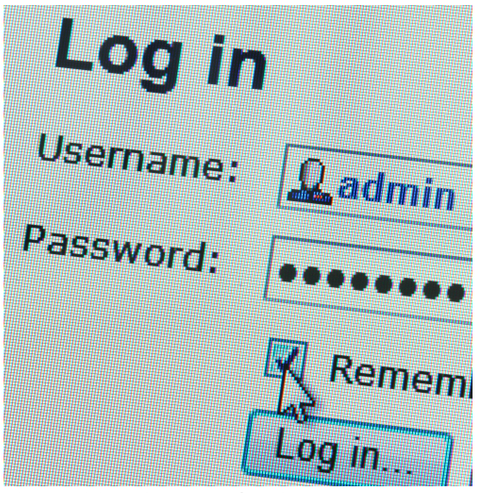


Figure 115: Password entry screen

This command will return the set of records that fit this query, which will either consist of one or zero records. Note that the text input from the user needs to be placed between inverted commas in the SQL command as the username is in the form of a string. If zero records are returned, the user either is not in the database or their password doesn't match. If one record is returned, the user has been authenticated.

One of the difficulties with this concerns the process of taking the direct input from the user and placing it into an SQL command such as the one above. If code is written in this way, the user has the ability to 'inject' their own code into the query. This is known as an SQL injection.

How does an SQL injection work?

A simple SQL injection involves adding an 'or' condition to the statement. Let's consider the example above again. The user could enter the following as their password:

```
password' or 1=1;
```

If this were to happen, the SQL name in the code would now look like:

```
SELECT * FROM users WHERE
        name = 'Bob' AND password = 'password'
        or 1=1;
```

The addition of the 'or 1=1' makes the statement always true. Not only will a statement like this potentially give the user access, it may give them access to the entire database, as the '1=1' will cause all of the usernames and passwords to be returned (as opposed to just one). A more insidious version of an SQL injection can be performed by placing additional SQL commands into the string. An example of this is shown below.

```
password'; DROP TABLE users;
```

The 'DROP TABLE' command deletes the entire table of that name. The user may not know the name of the database table they are trying to delete, but may attempt variations of common names till they hit upon one that works. In this instance, the user's intention is not to gain access, but rather, to cause intentional damage.

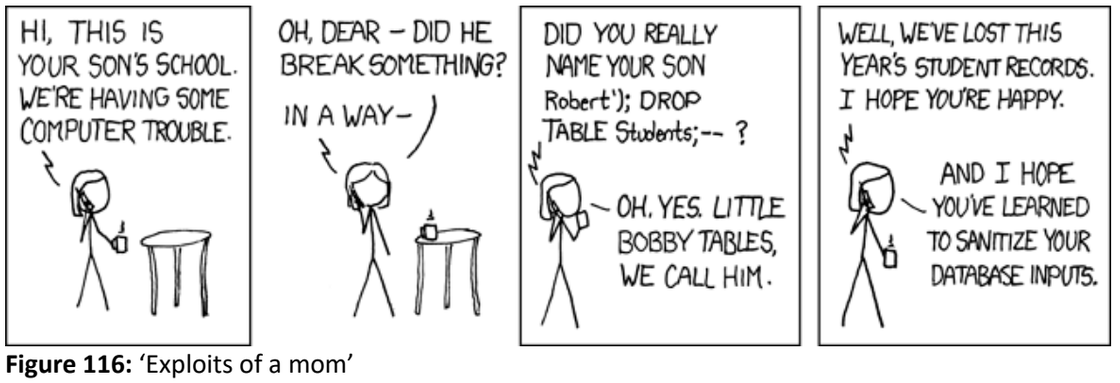


Figure 116: 'Exploits of a mom'
source: xkcd.com *used with permission*

Protecting from an SQL injection

Protecting from an SQL injection is relatively straightforward. By validating the inputs from the user prior to inserting them into an SQL string as parameters, a software developer can prevent giving the user the ability to enter code directly into the input. Despite the ease of this solution, a surprising number of information systems were vulnerable to SQL injections in recent years, and badly coded software solutions may still be.

A man in the middle attack

Another way an intruder can gain access to a user's data is to insert themselves in the middle of the communication that the user is having with the information system they are connected with. This generally involves eavesdropping on the initial communication so that the intruder can get the information that they need to make the connection, and then pretending to be the information system in future transactions with the user.

Unsecured wireless networks or wired networks in which users are connected are vulnerable to 'packet sniffing'. Any device on a network can 'listen' to the packets that are being transferred between network nodes.
'Packet sniffers' are not illegal by themselves as they can be important diagnostic tools. The use of a 'packet sniffer' can show how efficiently a network is operating and what bottlenecks are occurring.

The number of unsecured wireless connections that are offered to the public is growing. Coffee shops, shopping centres, hotels, airports and fast food restaurants have been offering connections of some sort for years, but many are now opening these up as the number of enabled devices grows. The attractions of using wireless hotspots of this kind are obvious and, although the connection will probably be sourced via a router with a firewall that protects you from some of the dangers inherent in connecting to the Internet, you are not protected from those that are closest to you. Ironically, the biggest threat to your data in this type of situation is not an unscrupulous hacker on the other side of the world but rather the (equally unscrupulous) one that is standing metres from you!

An unsecured connection is not only easy to connect to, it also means that the data being sent over it is unencrypted. The data that you send and receive is able to be 'sniffed'.

There are a number of settings within your operating system that can protect you however. The first of these is network discovery. Both Mac OS and Windows allow the user to turn network discovery off which basically means that your computer will not be seen on the network. In addition to this, it is important to disable all means of sharing files and resources – as often these are 'open' by default. When accessing any sites for which you will be entering sensitive information, you should use HTTPS rather than HTTP. HTTPS is an option that is available for most sites that

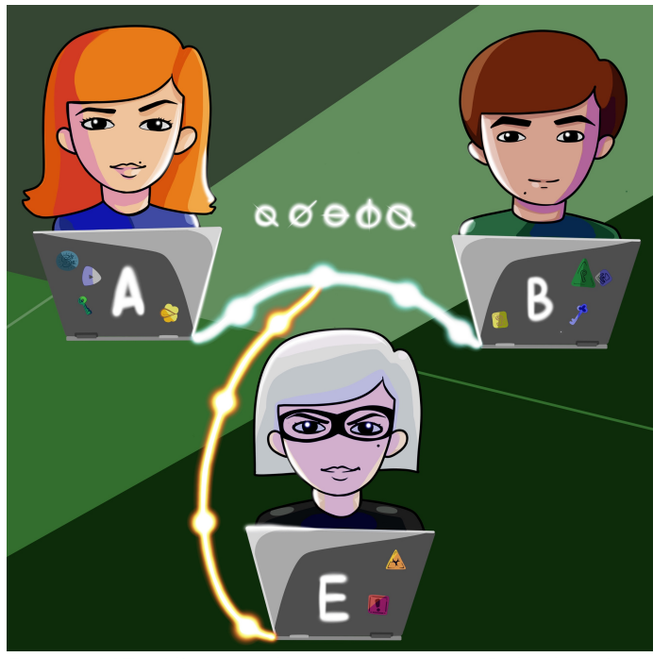


Figure 117: Alice sends a message to Bob, but it is intercepted by Eve (an eavesdropper).

require passwords or sensitive personal information and when accessing them an encrypted connection is established to protect your data (it is part of the TLS protocol discussed earlier). It is also a good idea to run your own software firewall.

A great way to establish a secure connection when connecting to an unsecure wireless access point is to connect to a Virtual Private Network (VPN). Once connected to a VPN (which could simply be your home computer), the sensitive information will be coming into the VPN (which is trusted) instead of straight into the unsecured wireless network. The connection to the VPN will, of course, be encrypted.

While this doesn't completely protect you from a 'man in the middle' attack, it does make things more difficult for the person attempting to do this.

Software security controls: The software and procedures used to assist in the protection of information systems and the files created, communicated and stored by individuals and organisations. These include usernames and passwords, access logs and audit trails, access restrictions, encryption, firewalls and system protection, and security protocols such as Transport Layer Security (TLS).

VCAA VCE IT Study Design 2020-2023 Glossary

Social engineering

In this chapter we have already discussed a wide range of physical and logical security measures that can be put in place to protect an information system. While an organisation can take a number of measures to protect their data, the phrase "a chain is as strong as its weakest link" is a relevant one to consider.

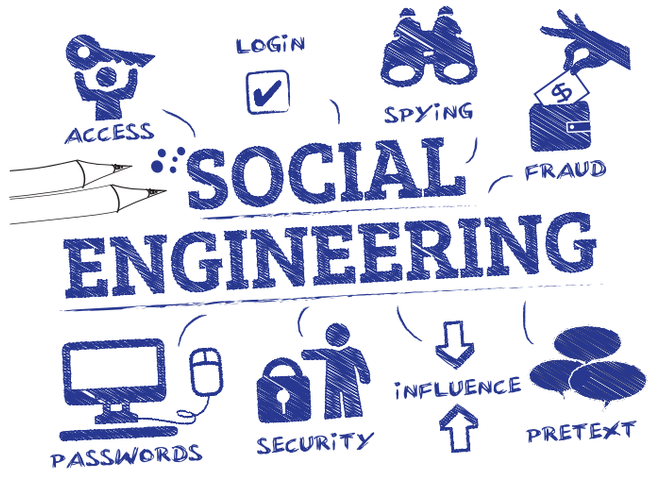


Figure 118: Some of the facets of social engineering

What many in the security industry also understand is, often this "weakest link" is "us".

Security systems can be set up and processes established so that these systems are used effectively, but if employees do not follow these procedures, security can be compromised.

Strictly speaking, the "art" of social engineering includes reading body language, understanding social cues, applying psychology and taking advantage of situations to leverage others to grant access to the system.

Security measures for the Internet

A number of security measures can be put into place to protect against many of these threats; there is not one magic solution and a multi-layered approach is required. Anti-virus software is a must. Anti-spyware software is also a necessary protection. Scripting is often an option that is available via web-browsers and other Internet enabled software packages and is a feature that can be disabled when 'surfing' in un-trusted zones. The use of a software firewall can be useful as an addition to a hardware one. Many security gurus recommend the use of most or all of these

protection methods as they complement each other.

On Windows computers, many users operate in administrator mode when they are working on day to day tasks. As the administrator account is only really needed for the installation of software, it should only be used when this needs to be done. The problem with working in administrator mode is that malware that infiltrates a computer that is logged on in this way, will also have administrator privileges. If malware tries to infiltrate a computer that is running in a 'user' mode, it will be unable to install or replicate itself in any way and the attack will be nullified. Many experts advocate only using administrator login when disconnected from the Internet entirely.

Keeping anti-virus and anti-spyware software up to date is essential as new malware threats are being created daily. There is some argument that keeping the operating system up to date is also a good practice, although some would also argue that it is good to wait before installing the latest operating system as sometimes security holes can present themselves and it can take a while for these to be detected and fixed.

Sending sensitive data via the Internet

Sending sensitive data over the Internet can only really be done by using encryption. Methods such as email, standard HTML web forms and the like are highly susceptible to 'sniffing', which is discussed in the context of Intranets later in this chapter.

Encryption

Encryption is the process of encoding information so that it is unreadable. This requires the use of an algorithm known as a cipher. In encryption, the original information is known as 'plaintext' and the encoded information is known as 'ciphertext'.

There are two main types of encryption: symmetric key encryption and asymmetric key encryption.

Symmetric key encryption

Symmetric key encryption can be used to send large amounts of information across the Internet from one person to another. The

In describing various cryptography and security examples, the characters of 'Alice' and 'Bob' have become the default for two users communicating with each other (A and B). Some other notable characters are:
Craig: A password cracker
Eve: An eavesdropper
Mallory: A malicious attacker
Trudy: An intruder
Sybil: An attacker that uses a number of different identities

'plaintext' version is encrypted using what is referred to as a 'secret key' and then sent to the recipient who then needs to decrypt it using the same key. This is a very secure way of sending information. The problem that arises is how does the sender let the recipient know what the secret key is? Obviously the key would not be included with the message carrying the 'ciphertext' as this would be too easy to intercept. It would also be as risky sending the key via the same transmission method either before or after the 'ciphertext'. If someone was indeed intercepting the messages, then it would be an easy matter for them to also intercept the key.

It may be that the secret key is exchanged successfully via another method that is secure. There is then the issue of how secure that key is over time. If the secret key is shared with other people, the security of the whole process diminishes as there are then many people who could potentially share the key or not keep it securely. A solution to this might be to create more secret keys and have keys that are used for communications with certain people,

however, this makes the process complicated and potentially confusing.

Asymmetric key encryption

In asymmetric key encryption, there are two keys instead of one – a public key and a private key. Both are related mathematically, but the public key can only be used to encrypt information and not decrypt it. The private key can be used to decrypt the information that has been encrypted by the public key.
This works in the following way. If someone wants to send information to someone else in a secure manner using asymmetric key encryption, they first request a copy of the public key. This enables them to encrypt the information and send it. When the information has been received, the owner of both the public and private keys is able to decrypt the information using the private key.

Public keys can be sent out to anyone who may wish to send encrypted data to you, but only you should have access to your private key.

Asymmetric key encryption works well but is not good for large amounts of information and is relatively slow compared to symmetric key encryption.

Secure Sockets Layer (or SSL) and Transport Layer Security (or TLS)

A common application of asymmetric key encryption is SSL (or Secure Sockets Layer) encryption, which establishes a secure 'handshake' between a web browser and a web-server. SSL is part of a security protocol called TLS (or Transport Layer Security).

TLS is commonly used when transferring sensitive information to a web-site (credit card information or personal details) and can also be used with email, VoIP, instant messaging and a variety of other applications. When accessing a secure web-site using this method, the user is sent the public key and the data is then encrypted and sent to the web-server. This link is usually indicated by a small padlock icon in the web-browser's title bar.

Hybrid encryption – a better solution

Symmetric and asymmetric key encryption have their pros and cons. While symmetric key encryption is very good at transferring large amounts of data quickly, it has problems with the exchange of the secret key. Asymmetric key encryption is slow, is one way only and has difficulties with large amounts of data. Both of these methods can be combined for a good solution that solves these problems.

Let's say two users (Alice and Bob) want to exchange information securely between them. Alice begins by making a transfer request of Bob who sends his public key to her. Alice then encrypts a secret key that she has generated for this transfer of information and he sends it to Bob. Bob is able to use his private key to decrypt the information that has been sent – which in this case is a secret key. Now that both Alice and Bob have access to the secret key, they can send information to each other securely and quickly.

Logical security

Logical security consists of software measures to protect access to the organisation's data and information. The best way to apply logical security measures is by establishing usernames, passwords and access restrictions (levels of access). Many organisations employ password policies that force their employees to enter complex passwords and change them frequently. Too often users will enter passwords that are easy for them to remember – but also easy to guess. The use of common

Figure 119: Security cameras

words as well as words or numbers that have meaning to them – such as family or pet names and birthdays, means that a person that wants to gain unauthorised access will often have some success by doing some research into the employees themselves. The prevalence of this sort of information on the Internet, especially via social networking sites such as Facebook, makes passwords that are built on these types of facts extremely weak. Many organisations now use password policies that force users to enter passwords of a minimum length, with a combination of upper and lower case characters, including at least one number and one special character (such as an exclamation mark, comma, hash or dollar sign).

Access logs and audit trails provide a useful layer of protection in that security breaches can, to some extent, be traced. In addition, if the users of a system know that access logs and audit trails are being produced, they might be less likely to attempt to access areas of the network to which they do not have access.

Physical security

Physical security measures prevent unauthorised access to physical hardware and software by using physical barriers and authentication techniques. The most obvious physical security measures (known as barrier techniques) consist of locks, alarms, fences and gates. While the use of these may seem to be quite obvious, their application within organisations can be inconsistent. Locks can be organised so that there are levels of physical access within an organisation (for example, some keys will access some areas). This is known as a zoned security strategy. Locks can also be digital locks that accept PINs or swipe cards. The advantage of this sort of system is that it can be installed in such a way that it maintains a log of when employees unlock or lock a particular door and the codes can easily be changed remotely. Alarm systems and video surveillance are both useful (and relatively cost effective), but their presence should be obvious so that they work both as a deterrent as well as a tool for detection and response.

Large organisations that can afford it will often employ guards or have 24-hour surveillance, but this is not an option that is always available and it comes down to a risk management (cost versus benefit). For many organisations however, the use of biometric security is quite cost effective and easy to implement.

Barrier techniques

> *Physical security controls: The equipment and procedures used to assist in the protection of information systems and the files created, communicated and stored by individuals and organisations. Equipment controls included zoned security strategies, barrier techniques and biometrics. Physical procedures include backing up, shredding confidential documents and checking authorisation credentials.*
>
> *VCAA VCE IT Study Design 2020-2023 Glossary*

The term 'barrier techniques' describes the control of access to an organisation through the use of barriers. Barriers are typically arranged to form concentric layers. The area of greatest sensitivity is at the centre of the layers and someone wishing to access this area must pass successfully through all of the layers in turn. Barrier techniques do not exclusively consist of physical security measures, although the first 'layers' can typically consist of fences, locks, guards, security cameras and gates.

Biometrics

The field of biometrics is concerned with identifying individuals based on unique identifiers of a physical or behavioural nature. Perhaps the best known biometric measure is a fingerprint. Fingerprint scanners can be placed on door locks or to provide access to computers or resources. Fingerprint scanners however, can be fooled easily with prosthetics or even photocopies of a fingerprint – provided that the fingerprint of an authorised person can be obtained. For this reason, they are considered a fairly low level biometric device.

Common physical biometric measures that are used are face recognition, palm prints, hand geometry and iris recognition. Examples of behavioural biometric measures that are used are gait, voice and typing rhythm. These measures can all be used to good effect, however, one of the best (and simplest) biometric measures is actually a photograph ID. For a photographic ID to be used, someone needs to physically check that the person in the ID photo is the person that wants to gain physical access to the organisation.

Figure 120: Fingerprint scanner

Tools and techniques for tracing transactions between users of information systems

We have already spoken about access logs and audit trails and how these provide a very useful method by which transactions between users of information systems can be traced. While it may not be desirable (or indeed ethical) to record every action that each user performs, a record of a transaction should be made when certain key events are triggered.

Transactions such as the following could be examples of events that should be recorded:

- Accessing a file
- Logging in and logging out of a system
- Printing a file
- Accessing a web-site

Threats to data integrity

Before we discuss the threats to data integrity, we must first define what it means for data to be labelled as 'having integrity'. For this to be true, a number of criteria need to be satisfied.

Accurate

Accurate data is data that is true to the source. It is captured (or entered) once and fully validated. Judging the accuracy of data can be a subjective thing, but there are a number of criteria that can be applied to assist in doing this.

- Does the data make sense in relation to the data and information that you have already gathered?
- Is the source of the information reputable and trustworthy?
- Has the data been checked?
- Is the data up to date?

To say that data is accurate is not that same as saying that it is correct. Accurate data is simply a true representation of the source material and it may actually be 'correct'.

Timely

All data has a 'use-by' date. Data that is considered timely for one purpose may not be useful for another. For example, data on the technology purchasing preferences of students will only be relevant for a short time as

technology is rapidly changing. The amount of time that can pass before data is no longer considered to be 'timely' really depends on what sort of data it is.

Reasonable

Data may be perfectly valid, but not reasonable in the context in which it is being used. For example, while it is possible for a person to live to the age of 100 (and beyond), it is not likely that a person of this age would be amongst a list of students taking part in an athletics carnival.

Authentic

The term 'authentic' relates to the source of the data. Is the data from a trusted source? The best way to ensure that data is authentic is to gather it yourself. If this is not possible, then to ensure authenticity, the person or the device gathering the data must be trusted and proven to work as expected. If the data is not being gathered first hand but is instead being collected from another source, then that source must be one that has a proven track record and transparent processes.

Correct

There are lots of terms to describe data and some of them are very similar. We have already spoken about validation, which is a process by which the data is made as error free as possible. Accuracy is a measure of how well data can be measured and recorded. We have also spoken about data that is authentic, however with data that is correct, the emphasis is on whether the data is right. Although it is unlikely, data could potentially be described as being compliant with all of the criteria mentioned previously, but still not be correct. To have a definitive answer as to whether data is correct or not, it needs to be compared or scrutinised by someone that has the knowledge or experience (or data from other sources) to be able to make that determination.

Poor data management, especially in relation to unstructured (or poorly structured) data, costs organisations a lot of money. The storage, control and transfer of data that is not in a structured format creates many challenges. Information systems within an organisation can easily lose or misplace data files or additional time may be needed to change the format of files to make them compatible with different systems. This can be as a direct result of a lack of standards within the organisation or potentially resulting from a lack of compliance in adhering to the standards that have been set.

The following are some other threats to data integrity.

Initial data conversion

Certainly once a database is established, well designed procedures and validation can keep the number of errors to a minimum. However, databases need to be created and populated with data and this process alone is one that can potentially introduce the most errors and inconsistencies. Some data may be entered that is completely new or represented in a different form. Other data may be converted from existing information systems.

There will certainly be lots of differences between these systems and the new databases being established: different fields types, data ranges and data that is missing. What will be done with data that exists in the existing information systems but not in the new system? What about gaps in the data or errors in the existing data?

Manual data entry

No matter how well designed an information system is, it will almost always require some manual data entry. With the intelligent design of user interfaces, many potential entry errors can be correctly validated and corrected. Despite this, there are some aspects of data entry that will always be prone to error no matter how well designed the entry process is. Data items such as names are easy to miss-

spell. It can be common for items to be miss-categorised (for example, 'male' as 'female'). Data items can be left out altogether or an element of one record entered into another.

Data mining

Data mining, by definition, is the process by which as much useful information is gleaned from a set (or sets) of data so that some deeper conclusions can be drawn. This can involve extracting and processing data from a variety of sources. The key question becomes one of trust. The more data that is included in a data set (especially data sets from a variety of sources), the more potential sources of error. When data sets are combined and processed to form new data, even though the new data set may be technically valid, the correctness of the original data then becomes hard to examine.

Incompatibility between systems

In any information system, databases or software packages will exchange data in a number of ways. These transactions can be one way or can be a true two-way exchange. In rare cases, an information system may have been built in an organisation and evolved so that it is the only source of data and it communicates with modules that are built following the same design. Given the time that it takes to build an information system and the changing needs of organisations, it is more likely to encounter a situation where many systems interact with each other.

Often a new software package will become available in the marketplace that will meet a need in an organisation. When this happens, one of the considerations is how the package will integrate with the existing systems. The integration may be a strong one or it may well be that data is exported from the main systems at regular intervals and then imported into the new software package using a batch process. While there is nothing wrong with doing this, the more transactions that are introduced, the more points of failure there will be in the process. What's more, there will likely need to be some translation between the existing systems and the new system.

It can be very easy in situations like this for the vendors of one software solution to 'blame' the inefficiencies on the legacy packages being used. Different data types could be used to represent the same data, which could create handling issues (a good example might be a postcode field being a numeric type in one system and a text type in another). Other items of data may be labelled differently (for example, 'Surname' versus 'Last Name') or there may be gaps in data sets (one software solution uses a 'Salutation' data field while another leaves this out).

User permissions

Another data management practice that can cause conflict between information systems could simply be that of user permissions. Systems administrators are often very careful with the permissions that they assign to those in the organisation, and rightly so. However, incorrectly set permissions could mean that a user is able to extract data from one system but is not then able to import the data into another to perform the task assigned to them. While this might seem like an easy issue to resolve, it could be the case that the permissions required by the user are not available as they would give the user access to many functions beyond their skill level or level of access.

Context Questions

1. What is the difference between a worm and a Trojan?
2. What is a DDOS attack and how can it be prevented?
3. What is the process known as 'packet sniffing'?
4. What is encryption and what does the simplest form of this involve?
5. What is the difference between logical security and physical security?
6. What does a complex password typically consist of?
7. When discussing physical security measures, what are barrier techniques?
8. What do biometric devices measure?
9. When an attack occurs on an organisation, what information can be gained and how?
10. What is penetration testing and who is it carried out by?

Applying the Concepts

- Interview the network manager at your school and ask them to describe the physical and software controls that they use to secure their data.
- Investigate cases where a loss of data integrity has led to a significant (and possibly detrimental) effect for customers of a business. Be sure to focus on cases that are specifically due to a loss of data integrity as opposed to the effects of viruses or errors in code.
- Discuss 'best practice' for creating passwords. Make a list of guidelines that can be followed by users to ensure they do everything they can to protect their data.
- Discuss times in which you or others that you know have changed the outcomes of situations by appealing to the nature of the other parties or by "stretching the truth". This needs to be a 'safe' discussion, but can serve to highlight how social engineering can be used and how it can be guarded against.

Key Skills Checklist

At the conclusion of this chapter, you should be able to address the following key skills. Mark each off as you can achieve them.

Understand the difference between physical and software controls to protect data ☐

List and describe strategies that can be used to identify and minimise risks ☐

Describe types of risks and the ways that they can be protected against ☐

List and describe the characteristics of data that has integrity ☐

Describe how ineffective security strategies can have an impact on integrity of data ☐

Understand how to formulate criteria to assess the effectiveness of security strategies ☐

Discuss risk management strategies that can be used to minimise security vulnerabilities ☐

Sample Examination Questions

The following sample examination questions can be attempted to test your knowledge of the content of this chapter.

"Even though I created this module for my last job, it's mine to sell and do what I wish with".

Question 1
Which of the following would you classify as an event based threat?

- **A.** Incorrect file formats leading to files being deleted
- **B.** Loss of hardware devices
- **C.** Water damage due to flood
- **D.** Viruses damaging data

Question 2
One of the functions of a firewall is:

- **A.** To prevent unauthorised users from accessing the network
- **B.** To ensure that the Internet is protected from viruses on the internal network
- **C.** To provide a method of logging people in
- **D.** To run network software

Question 3
Which of the following is an example of a logical security measure?

- **A.** Biometrics
- **B.** Security cameras
- **C.** Passwords and levels of access
- **D.** Doors with swipe card locks installed

Question 4
An example of an accidental threat to data could be:

- **A.** Flood in the basement of the building
- **B.** Unauthorised access to the information system
- **C.** Users not familiar with how to use the system properly
- **D.** Power failure

Question 5
Employees working for a loan financing company have been given tablets that have WiFi and 4G network access and connect to the company's servers to access data while quotes are being given to customers in their own homes.

a. List **three** threats to these devices.

1: _____

2: _____

3: _____

3 marks

b. Jill (Client Services Manager) is in favour of locking the devices with a common password (such as 'ABC123') while Kalen (IT Support) wants to use the build in finger-print scanner that the devices have. Discuss the merits of each approach.

_____

_____

_____

2 marks

Question 6

One of the most common sources of accidental threats are the employees of a company. Describe **two** ways that this threat can be minimised.

1: _____

_____

_____

2: _____

_____

_____

2 marks

Question 7

Beans Meanz is a small suburban café that offers free WiFi to anyone in the area. For convenience, it has been left "open" without any security or encryption on it at all. Traci, who has only recently been hired to work at the café, has some network security experience and has been critical of this arrangement.

a. What is the difference between a network password and encryption?

_____

_____

2 marks

b. List **two** reasons why Traci might be critical of the setup at *Beans Meanz*.

Reason 1: _____

_____

Reason 2: _____

_____

2 marks

Question 8
Most organisations employ both logical and physical security controls to protect their information systems. Describe **two** common logical security measures and **two** common physical security measures that can be used.

Logical measure 1: _____

_____

Logical measure 2: _____

_____

Physical measure 1: _____

_____

Physical measure 2: _____

_____

4 marks

Sample Examination Answers

Question 1
Answer: C

Question 2
Answer: A

Question 3
Answer: C

Question 4
Answer: C

Question 5

a. 1: theft of the device.
   2: damage to the device (dropping, impact, etc)
   3: breach of the company server / database via unauthorised access
b. Using a common password will mean that the employees on the road will be easily able to remember how to access the device and will be able to ask each other if they forget. The password that has been chosen however, is a very simply one to crack or for someone to work out if they are watching the employee enter it.
   A fingerprint scanner is much more secure and will ensure that the device is only being accessed by an authorised employee. There may be some circumstances in which it may not be practical (injury, wet or dirty hands).

Question 6

1: Backup data frequently.
2:Train staff in the correct procedures and use of software packages.

Could also maintain security levels of access and audit trails, so that changes can only be made by those with access and even then, can be traced and rolled back.

Question 7

a. A network password will gain you access to a network. Encryption is a method by which the data that is transferred is encoded so that it can't be read (without the correct key).
b. Reason 1: Anyone could access resources on the network and implant viruses or copy (or delete data).
   Reason 2: Data transmitted over a WiFi network can be easily read by others if it is not encrypted. This could include banking details and passwords, for example.

Question 8

Logical measures

- Usernames and passwords so that only those that are authorised can gain access to the information system.
- Levels of access which only allow those with the correct clearance in an organisation to access important / critical files.

Physical measures

- Security cameras which monitor critical areas such as the server room.
- Swipe cards on doors so that only those with clearance can gain physical access to certain areas.

Other possible answers include the use of logs / audit trails and biometric devices such as fingerprint scanners / locks.