LEVEL 3 ALTERNATIVE ACADEMIC QUALIFICATION
CAMBRIDGE ADVANCED NATIONAL

Computing: Application Development

Student Book

Chris Charles, David Corbett & Jen Gainsford

Shaftesbury Road, Cambridge CB2 8EA, United Kingdom

One Liberty Plaza, 20th Floor, New York, NY 10006, USA

477 Williamstown Road, Port Melbourne, VIC 3207, Australia

314–321, 3rd Floor, Plot 3, Splendor Forum, Jasola District Centre, New Delhi – 110025, India

103 Penang Road, #05–06/07, Visioncrest Commercial, Singapore 238467

Cambridge University Press & Assessment is a department of the University of Cambridge.

We share the University's mission to contribute to society through the pursuit of education, learning and research at the highest international levels of excellence.

www.cambridge.org
Information on this title: www.cambridge.org/9781009817134

First published 2025
20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

Printed in Great Britain by Ashford Colour Press Ltd.

A catalogue record for this publication is available from the British Library

ISBN 978-1-00-981713-4 Paperback
ISBN 978-1-00-981712-7 Digital Edition
ISBN 978-1-00-981715-8 Digital Edition (1 Year Site Licence)

Additional resources for this publication at www.cambridge.org/9781009817134

Contents

Acknowledgements

The authors and publishers acknowledge the following sources of copyright material and are grateful for the permissions granted. While every effort has been made, it has not always been possible to identify the sources of all the material used, or to trace all copyright holders. If any omissions are brought to our notice, we will be happy to include the appropriate acknowledgements on reprinting.

Thanks to the following for permission to reproduce images:

Cover oxygen/Getty Images

Inside **F160** Quantic69/GI; **TA1** Igor Zhukov/GI; Booka1/GI; Daniel Balakov/GI; **TA2** Diki Prayogo/GI; Flavia Morlachetti/GI; **TA3** AndreyPopov/GI; Artvea/GI; **TA5** Yurich84/GI; Merovingian/GI; Capuski/GI; Oscar Wong/GI; Courtneyk/GI; SpiffyJ/GI (x2); **TA6** Xavierarnau/GI; Cavan Images/GI; Portra/GI; **F161** SpicyTruffel/GI; **TA1** Gorodenkoff/GI; Mrgao/GI; Pictafolio/GI; IndiaPix/IndiaPicture/GI; Sitthiphong/GI; Yevgen Romanenko/GI (x2); Figure 2.11 Photography by Jonathan Banks for Microsoft; **TA4** Fotoscape/GI; MirageC/GI; Douglas Sacha/GI; Oscar Wong/GI; **TA5**: Ole_CNX/GI; **TA6** Sean Gladwell/GI; **F162** Korawat thatinchan/GI; **TA1** SFL Travel/ Alamy Stock Photo; Bob Riha Jr/GI; New York Daily News Archive/GI; NurPhoto/GI; Alex Ruhl/Alamy Stock Photo; **TA2** Oscar Wong/GI; CarmenMurillo/GI; Tomas Knopp/GI; Trinetuzun/GI; NicoElNino/GI; Amenic181/GI; Rawpixel/GI; **TA3** Bagi1998/GI; Figure 3.27 Reproduced with permission of Jacob Farney; **TA4** Courtneyk/GI; Figure 3.33 By permission of IntentUK.com; Sesame/GI (x2)

Key: GI = Getty Images

Adobe product screenshots reprinted with permission from Adobe.

Introduction

About your course

Working in the computing industry is very rewarding. The Level 3 Alternative Academic Qualification Cambridge Advanced National in Computing: Application Development will help you to develop key knowledge, understanding and skills that are important for jobs and careers within the sector. During the course you will be encouraged to:

- think creatively, innovatively, analytically, logically and critically
- develop valuable communication skills
- develop transferable learning and skills, such as evaluation, planning, presentation and research skills
- develop independence and confidence in applying the knowledge and skills that are vital for progression to higher education and work situations.

How you will be assessed

You can take the Level 3 Alternative Academic Qualification Cambridge Advanced National in Computing: Application Development at one of two levels:

Certificate

For this qualification, you must complete two units:

- One mandatory externally assessed unit (E)
- One mandatory non-examined assessment unit (NEA).

Unit number	Unit title	How is it assessed?	Mandatory or optional
F160	Fundamentals of application development	E	M
F162	Designing and communicating UX/UI solutions	NEA	M

Extended Certificate

For this qualification, you must complete five units:

- Two mandatory externally assessed units
- One mandatory NEA unit
- Two optional NEA units.

Unit number	Unit title	How is it assessed?	Mandatory or optional
F160	Fundamentals of application development	E	M
F161	Developing application software	E	M
F162	Designing and communicating UX/UI solutions	NEA	M
F163	Game development	NEA	O
F164	Website development	NEA	O
F165	Immersive technology solution development	NEA	O
F166	Software development	NEA	O

The **Level 3 Alternative Academic Qualification Cambridge Advanced National Computing: Application Development Student Book** will support you with your Mandatory units. Support for the Optional units can be found at **cambridge.org/cambridge-advanced-nationals**.

About the authors

Chris Charles

Chris Charles is an experienced teacher, author, educational consultant and head of digital learning. His published material includes books, schemes of work, online revision material and CPD courses. He supports a number of organisations with technical and pedagogical knowledge. Chris also works on a Subject Knowledge Enhancement programme and with trainee teachers to secure their subject knowledge. Chris is interested in promoting high-quality digital learning in all subject areas.

David Corbett

David Corbett is an experienced teacher. He graduated from the University of Wales, Aberystwyth. Beyond his classroom responsibilities, David has expanded his influence within the educational community by becoming an educational resource provider for secondary schools across the UK. He has developed and provided a diverse range of resources tailored to various courses in both England and Wales. Additionally, he has offered bespoke training sessions focused on the teaching and assessment of the Cambridge Nationals Creative iMedia course, supporting teachers in their delivery of the curriculum.

Jen Gainsford

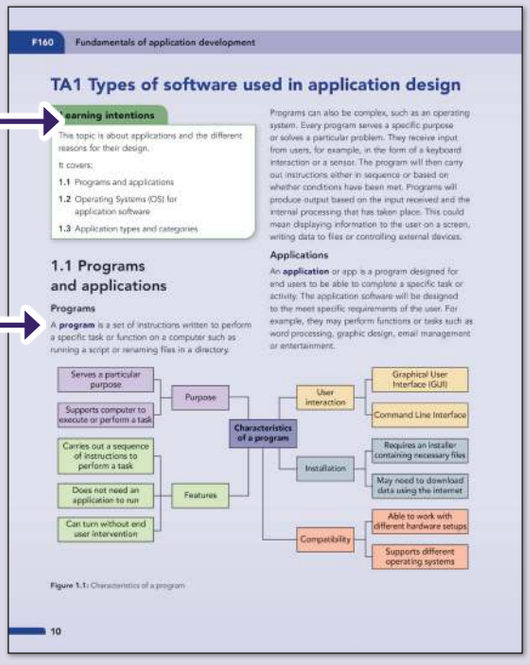
Jen Gainsford is an experienced Head of Department who taught a range of different computing courses across the 11–18 age range during her 15 years of teaching. Since leaving teaching in 2023, Jen has worked to build digital capability and skills within a higher education institution, but still works as a trainer, examiner and moderator in her spare time. Jen has also previously authored a range of teacher resources and student textbooks.

How to use this book

Throughout this book, you will notice lots of different features that will help your learning. These are explained below.

Learning intentions – each topic begins with a set of learning intentions to show you what the topic covers.

Key words – key words are highlighted in the text and explained fully in the glossary to ensure you understand key terminology.


F160 Fundamentals of application development

TA1 Types of software used in application design

Learning intentions

This topic is about applications and the different reasons for their design.

It covers:

1.1 Programs and applications

1.2 Operating Systems (OS) for application software

1.3 Application types and categories

1.1 Programs and applications

Programs

A **program** is a set of instructions written to perform a specific task or function on a computer such as running a script or renaming files in a directory.

Programs can also be complex, such as an operating system. Every program serves a specific purpose or solves a particular problem. They receive input from users, for example, in the form of a keyboard interaction or a sensor. The program will then carry out instructions either in sequence or based on whether conditions have been met. Programs will produce output based on the input received and the internal processing that has taken place. This could mean displaying information to the user on a screen, writing data to files or controlling external devices.

Applications

An **application** or app is a program designed for end users to be able to complete a specific task or activity. The application software will be designed to the meet specific requirements of the user. For example, they may perform functions or tasks such as word processing, graphic design, email management or entertainment.

Characteristics of a program

Purpose: Serves a particular purpose; Supports computer to execute or perform a task

Features: Carries out a sequence of instructions to perform a task; Does not need an application to run; Can turn without end user intervention

User interaction: Graphical User Interface (GUI); Command Line Interface

Installation: Requires an installer containing necessary files; May need to download data using the internet

Compatibility: Able to work with different hardware setups; Supports different operating systems

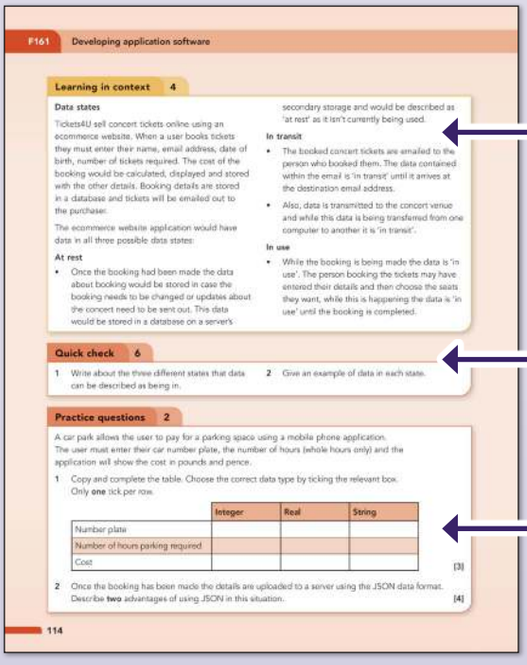
Figure 1.1: Characteristics of a program

10

Learning in context – this puts key concepts and practices into every day and working life, in the form of realistic and industry-relevant examples.

Quick check questions – review-style questions at the end of each sub-topic enable you to check your knowledge and understanding.

Practice questions – these questions appear at the end of each topic in Unit F160 and F161. They give you the opportunity to test your knowledge and understanding to help you to prepare for examination.


F161 Developing application software

Learning in context 4

Data states

Tickets4U sell concert tickets online using an ecommerce website. When a user books tickets they must enter their name, email address, date of birth, number of tickets required. The cost of the booking would be calculated, displayed and stored with the other details. Booking details are stored in a database and tickets will be emailed out to the purchaser.

The ecommerce website application would have data in all three possible data states:

At rest

- Once the booking had been made the data about booking would be stored in case the booking needs to be changed or updates about the concert need to be sent out. This data would be stored in a database on a server's secondary storage and would be described as 'at rest' as it isn't currently being used.

In transit

- The booked concert tickets are emailed to the person who booked them. The data contained within the email is 'in transit' until it arrives at the destination email address.
- Also, data is transmitted to the concert venue and while this data is being transferred from one computer to another it is 'in transit'.

In use

- While the booking is being made the data is 'in use'. The person booking the tickets may have entered their details and then choose the seats they want, while this is happening the data is 'in use' until the booking is completed.

Quick check 6

1 Write about the three different states that data can be described as being in.

2 Give an example of data in each state.

Practice questions 2

A car park allows the user to pay for a parking space using a mobile phone application. The user must enter their car number plate, the number of hours (whole hours only) and the application will show the cost in pounds and pence.

1 Copy and complete the table. Choose the correct data type by ticking the relevant box. Only **one** tick per row.

	Integer	Real	String
Number plate			
Number of hours parking required			
Cost			

[3]

2 Once the booking has been made the details are uploaded to a server using the JSON data format. Describe **two** advantages of using JSON in this situation. [4]

114

F160 Fundamentals of application development

Topic Areas

TA1: Types of software used in application design

TA2: Software development models

TA3: Planning application development projects

TA4: Application design scoping

TA5: Human computer interface and interaction

TA6: Job roles and skills

In today's digital age, applications play a crucial role in various aspects of our lives, from communication and productivity to entertainment and education. Understanding the fundamentals of application development is essential for anyone aspiring to become a proficient software developer or designer.

In this unit you will learn about applications and their functions, reflecting on how client requirements can shape the software choices that are made. You will learn about the different processes that take place when designing an application, from initial planning to exploring design ideas and features. You will also learn about the variety of job roles in this area, understanding how each contributes to application development and their main roles and responsibilities.

TA1 Types of software used in application design

Learning intentions

This topic is about applications and the different reasons for their design.

It covers:

1.1 Programs and applications

1.2 Operating Systems (OS) for application software

1.3 Application types and categories

1.1 Programs and applications

Programs

A **program** is a set of instructions written to perform a specific task or function on a computer such as running a script or renaming files in a directory.

Programs can also be complex, such as an operating system. Every program serves a specific purpose or solves a particular problem. They receive input from users, for example, in the form of a keyboard interaction or a sensor. The program will then carry out instructions either in sequence or based on whether conditions have been met. Programs will produce output based on the input received and the internal processing that has taken place. This could mean displaying information to the user on a screen, writing data to files or controlling external devices.

Applications

An **application** or app is a program designed for end users to be able to complete a specific task or activity. The application software will be designed to the meet specific requirements of the user. For example, they may perform functions or tasks such as word processing, graphic design, email management or entertainment.

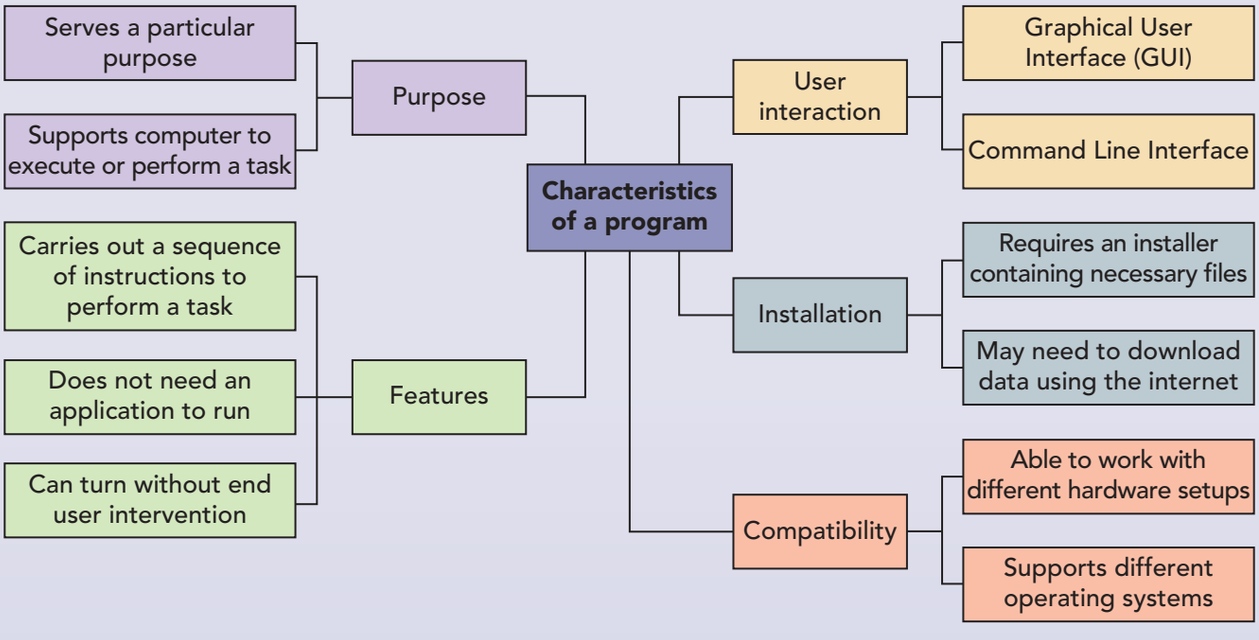


Figure 1.1: Characteristics of a program

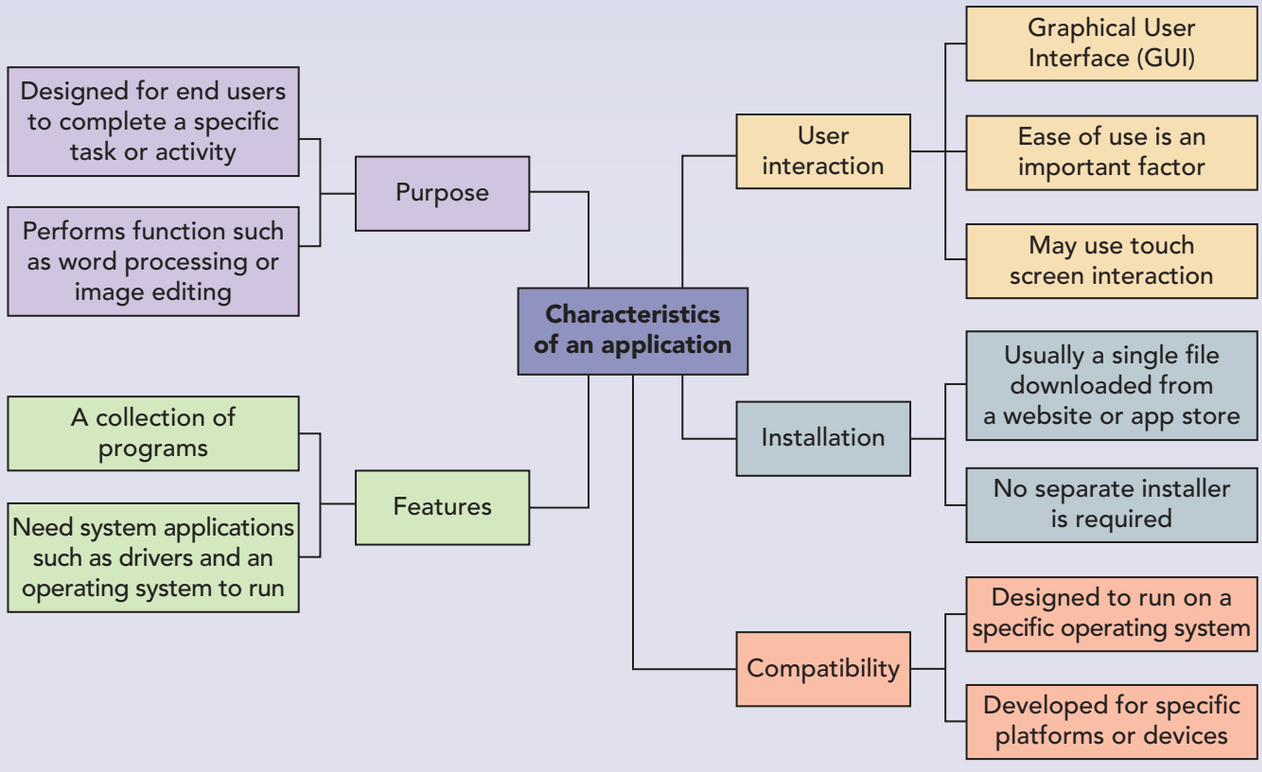


Figure 1.2: Characteristics of an application

Applications are developed to run on a specific operating system, utilising the operating system and other system software to function such as the file storage system. They will also be developed for specific platform or device types. An Android mobile application for an Android phone, for example, will be different to a Windows application designed to run on a desktop PC. Multiple versions of the same application may be developed to meet these differing needs.

Applications generally have a user-friendly Graphical User Interface (GUI) to make them easier to use. They require installation and can be downloaded from an app store or website.

Applications are essentially a type of program. However, not all programs are applications. Programs include a broader category of software including both applications and other types of software such as system utilities, drivers and scripts.

There are some good examples of devices using programs and application software highlighted in Table 1.1.

Table 1.1: Programs and applications can be found on a variety of different devices

Device	Example
Personal computers (PCs)	Desktop computers, laptops and tablet PCs are devices that run a variety of programs and applications. They are used for tasks such as word processing, web browsing, multimedia, gaming and software development.
Smartphones and tablets	Smartphones and tablets are mobile computing devices equipped with powerful processors and operating systems that support a wide range of applications. These devices are used for communication, social media, web browsing, email, productivity tasks, gaming, navigation and more.
Servers	Servers are powerful computers designed to provide services and resources to other devices on a network. They run specialised programs and applications, such as web servers, email servers, database servers and file servers.
Embedded systems	Embedded systems are specialised computing devices integrated into larger systems or products to perform specific functions. They often run programs and applications tailored to their intended purpose, such as industrial control systems, medical devices, automotive systems and consumer electronics.
Wearable devices	Wearable devices, including smartwatches, fitness trackers, Augmented Reality (AR) glasses and health monitoring devices, run specialised programs and applications that track fitness metrics, provide notifications and enable interactive experiences.
Gaming consoles	Gaming consoles are dedicated gaming devices that run programs and applications tailored for gaming experiences. These devices support a variety of gaming software, including console-exclusive titles, online multiplayer games and multimedia entertainment apps.
Smart home devices	Smart home devices, such as smart speakers, smart thermostats, smart locks, and smart appliances, incorporate programs and applications to automate tasks, control home functions remotely and interact with other smart devices in a connected home ecosystem.

Quick check 1

1 Discuss with a partner the similarities and differences between programs and applications.

2 Think about an application you are familiar with. Make notes on how the application meets the needs of its end user.

1.2 Operating Systems (OS) for application software

Network

Network operating systems are specifically tailored to support the functionalities of a network. The typical functions of a network operating system include:

- resource sharing across the network
- user authentication and access control
- providing communication services between networked devices
- network administration tools.

An example of a network operating system is Windows Server, Microsoft's server operating system which is widely used in enterprise environments.

Network operating systems are found in environments where there are multiple computers and devices that are connected to form a network. These devices might include:

- a server, the central computer that provides various services and resources to other computers
- workstations, individual computers that are used by users to perform tasks such as web browsing and document editing
- printers, scanners and storage devices.

Table 1.2: Advantages and disadvantages of network operating systems

Advantages	Disadvantages
• System updates can be easily rolled out across the network • Central server provides stability • Enhanced security such as user authentication.	• Initial costs for a central server, hardware and software • Ongoing costs for regular maintenance and support • Hardware and software can have compatibility issues with network operating systems.

Open OS

An **open operating system** is a software system whose source code is freely available for anyone to use, modify, and distribute under an agreed set of rules called a licence. Open operating systems encourage collaboration between users to create a system to meet their needs. Examples include Linux based systems such as Ubuntu.

Table 1.3: Advantages and disadvantages of open operating systems

Advantages	Disadvantages
• Users and developers can modify and customise the system to meet their needs • Community support, such as forums, makes it easy to access help and guidance • Many open systems are free or at a low cost compared to alternatives.	• Operating systems can be complex and need a level of technical expertise to configure, customise and maintain them • Compatibility with hardware devices or different versions of the operating system can be an issue.

Open operating systems are widely adopted across various industries and applications.

- Smartphones and tablets using the Android operating system, developed by Google offers a customisable user interface and support for a wide range of hardware configurations. Through the Google Play Store there is an extensive app ecosystem available for developers.
- Computers and laptops running Linux-based operating systems such as Ubuntu offer a customisable and secure computing environment. They are widely used in desktops, laptops, and servers with a vast selection of software packages available.
- Single-board computers such as a Raspberry Pi run on open-source operating systems providing users with a platform for learning programming and experimenting with electronics.

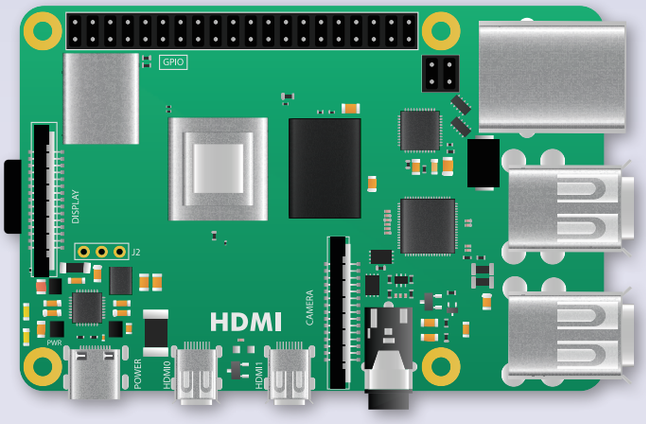


Figure 1.3: Photograph of a Raspberry Pi model B board

Proprietary

A **proprietary operating system** is a software system developed and owned by a specific company or organisation. The source code is not publicly available, and the use, modification and distribution of it is not allowed. This type of system is developed to generate revenue through fees paid for a licence to use it or associated hardware. Examples of proprietary operating systems include Microsoft Windows, macOS and iOS.

There are a wide range of devices that rely on proprietary operating systems. Some common examples of devices using proprietary systems include:

- Smartphones and tablets using Apple's iOS, which is exclusive to Apple mobile devices such as iPhones and iPads. It has a sophisticated graphical user interface and seamless integration with other Apple-based products.
- Computers and laptops, for example MacBooks and iMacs, use Apple's macOS proprietary operating system. It offers a user-friendly interface, robust built-in applications, and seamless integration with other Apple devices. By contrast, a Microsoft PC or laptop would run a Windows-based operating system.
- Gaming consoles such as Sony's PlayStation and Microsoft's Xbox consoles use proprietary operating systems tailored for gaming and entertainment experiences. These include features to enable online gaming, media streaming and access to online stores.

Table 1.4: Advantages and disadvantages of proprietary operating systems

Advantages	Disadvantages
• Often prioritises ease of use and user-friendly interfaces, making them accessible to a wide range of users, including those with limited technical expertise • Include comprehensive support options to provide help for users • Optimised for specific hardware configurations, resulting in better performance and compatibility with supported devices.	• Opportunities for customisation are limited to the restrictions imposed by the company • Licencing fees or subscription costs can increase total cost of ownership and impact affordability of the system • May be more susceptible to security vulnerabilities due to the closed nature of development and limited transparency.

Operating systems and client requirements

Choosing an operating system on which to develop an application is important as it affects the reach, user experience and technical requirements of the application. Considerations might include:

- Compatibility: the most crucial factor is ensuring that the chosen operating system supports the application's required features and functions. Some applications are designed specifically for certain operating systems or may have dependencies that require a particular version.
- Performance: if an application requires real-time processing or high computational power, the operating system must be capable of efficiently managing system resources, such as the CPU and memory.

- Security: applications that deal with sensitive data or operate in regulated industries may need to have an operating system that has robust security features and a track record of addressing vulnerabilities quickly.
- User experience: different operating systems have distinct user interfaces, mechanisms of interaction and customisation options. The chosen operating system should align with the target users' preferences and provide an intuitive and seamless user experience.
- Licencing and costs: some operating systems are open-source and free to use, while others require purchasing licences or subscriptions. Clients may want cost-effective solutions that meet the application's requirements without creating unnecessary expenses.
- Support and maintenance: the availability of support and maintenance services for the chosen operating system is crucial for ensuring the application's reliability and stability. Operating systems with a large user base often have extensive documentation, community forums and professional support options available.

Learning in context 1

Selecting an operating system

Pixel Logic Inc. is a software development company specialising in image editing and enhancement applications. With a focus on providing intuitive and powerful tools for both amateur and professional photographers, Pixel Logic Inc. aims to create innovative solutions that empower users to unleash their creativity.

Pixel Logic Inc. has decided to develop a new photo editing application to meet the growing demand for versatile and user-friendly image editing tools. One of the key decisions facing the company is choosing the appropriate operating system(s) to develop the application for. The management team is debating between iOS, Android, and Windows as potential platforms for these applications are the mostly commonly used operating systems on mobile devices.

Table 1.5: Factors considered by Pixel Logic when making their choice

Factor	iOS	Android	Windows
Market share and user base	Known for its affluent user base and strong presence in the creative industry. Apple devices, such as iPhones and iPads, are popular among photographers and designers.	Dominates the global smartphone market with a wide user base, offering potential for reaching a large audience.	Less prevalent in the mobile space compared to iOS and Android but has a presence in the desktop and laptop markets, particularly among professionals and businesses.
Development environment	Development for iOS requires specialist coding knowledge. Apple's strict guidelines ensure consistency and quality across iOS apps.	The platform offers greater flexibility but may present challenges due to the range of different devices using this platform and compatibility issues.	Development for Windows can be done using Visual Studio and languages like C# or C++. Allows for building applications that run across multiple Windows devices.

(Continued)

Learning in context Continued

Factor	iOS	Android	Windows
Monetisation strategy	The App Store is known for higher user spending, offering potential for generating revenue through app sales and in-app purchases.	Google Play Store has a larger number of free apps and relies more on ad-based monetisation models. In-app purchases and subscriptions are also common.	The Microsoft Store provides opportunities for app distribution, but the user base may be smaller compared to iOS and Android.

Quick check 2

1 Make notes on the key differences between open and proprietary operating systems.

2 Imagine you want to create a new application for a desktop computer. What operating systems could you choose from? Consider the pros and cons of each.

1.3 Application types and categories

1.3.1 Application types

Figure 1.4: These types of file are created and edited using applications

Communication

Communication applications are used to enable communication between users through various means such as text, voice, video calls and messaging. Characteristics include user-friendly interfaces, support for multimedia sharing, encryption for secure communication, integration with social media platforms and presence of features like group chats and video conferencing.

Educational

Educational applications are used to aid learning and increase knowledge, often through interactive lessons, quizzes, tutorials and educational resources. Characteristics include structured learning content, progress tracking, use of games and access to a wide range of subjects and topics.

Entertainment

Entertainment applications provide leisure activities and content for enjoyment and relaxation. Characteristics include content such as movies, TV shows, music, podcasts and eBooks with personalised recommendations, social sharing features, ability to download content for later. They may also use subscription-based models.

Games

Game applications give interactive experiences for entertainment and enjoyment, often involving challenges, strategy and competition. Characteristics include a huge range of game genres including

action, adventure, puzzle, simulation and role-playing games. They also include engaging gameplay mechanics, high-quality graphics and sound effects, multiplayer options, in-app purchases and regular updates.

Lifestyle

Lifestyle applications assist users in managing various aspects of their daily lives, including health, fitness, finance, travel and personal organisation. Characteristics include features tailored to specific lifestyle needs such as workout tracking, budget management, travel planning tools, habit-building functions, reminders and customisation options.

Productivity

Productivity applications help users enhance efficiency and accomplish tasks effectively, both professionally and personally. Characteristics include tools for task management, note-taking, document editing, calendar integration, collaboration features, cloud storage synchronisation, cross-platform compatibility and support for integrations with other productivity tools.

Protection and utility

Protection and utility applications ensure the security, privacy and smooth functioning of devices by offering various utility functions and security features. Characteristics include antivirus and malware protection, device optimisation tools, data backup and recovery options, privacy controls, password managers, VPN services and system monitoring functionalities.

Web browsers

Web browsers enable users to access and navigate the World Wide Web, view web pages and interact with online content. Characteristics include a fast and responsive browsing experience, support for multiple tabs, bookmarking, private browsing modes, synchronisation across devices, compatibility with web standards and technologies and extensions or add ons for additional functionality.

1.3.2 Application software categories

Open

Open application software are applications that are built using open-source principles and are freely available for anyone to use, modify and distribute. They often encourage collaboration among developers and communities, leading to rapid innovation and improvement. The purpose of open application software is to provide accessible, flexible, and collaborative solutions that empower users and promote innovation in the software development ecosystem. Examples include VLC Player, Audacity and Mozilla Firefox.

Table 1.6: Advantages and disadvantages of open application software

Advantages	Disadvantages
• Flexibility: allow for a high degree of customisation and flexibility. Users can tailor applications to their specific needs, creating functionalities that may not be available in standard, closed applications. • Innovation: encourage innovation as they allow developers and users to experiment with new ideas, features and integrations. • Community collaboration: enable collaboration within developer communities. Developers can share their work, collaborate on projects and learn from each other.	• Quality control: as anyone can contribute, there may be a wide variation in the quality of applications, with some being poorly designed, insecure, or unreliable. • Security risks: if developers do not follow best practices or adequately test their code, there can be security risks that can potentially compromise the security of an entire system or network. • Lack of support: users may have to rely on community forums or self-help resources for assistance, which may not always be comprehensive enough.

Devices that use open application software include:

- personal computers such as desktops and laptops
- servers
- mobile devices such as smartphones and tablets
- embedded systems such as routers
- industrial equipment such as automation systems and robotics
- educational devices such as interactive whiteboards, educational robots.

Closed

Closed software is proprietary software developed and distributed by companies or organisations for commercial purposes. Users typically do not have access to the source code and modifications are often not allowed. The main purpose is to generate revenue through the sale of licences or subscriptions. Examples include Microsoft Office, Adobe Photoshop and AutoCAD.

Closed-source applications are commonly found in:

- personal computers such as desktops and laptops
- embedded systems such as consumer electronics and medical devices.

Shareware

Shareware software is distributed on a trial basis, allowing users to evaluate its features before purchasing a licence. Typically, shareware versions have limited features or a trial period after which users are prompted to purchase the full version. The main purpose of shareware is to encourage users to try out the software and then purchase the full version if they find it useful. It's a marketing strategy often used by independent developers or small software companies. Examples include WinRAR and WinZip.

Shareware applications are typically found in:

- Personal computers such as desktops and laptops
- Mobile devices such as smartphones and tablets.

Freeware

Freeware software is available for use at no cost. Users can download, install and use freeware without any payment required. Freeware is often distributed with an open license, allowing users to change and sell the software. Freeware is typically developed by individuals, organisations or communities for various purposes, including educational or recreational reasons. While it is offered for free, it serves a purpose for developers who may still benefit from donations, advertising revenue or by using it as a promotional tool for other products or services. Examples include VLC Media Player, Audacity and Firefox.

Freeware applications are commonly found in:

- personal computers such as desktops and laptops
- mobile devices such as smartphones and tablets
- consumer electronics such as smart TVs and digital cameras.

Embedded

Embedded software is specifically designed to perform dedicated functions within a larger system or device. It's often tightly integrated with hardware and may have limited user interaction or visibility. Embedded software is used in a wide range of devices and systems, including consumer electronics, automotive systems, medical devices, industrial equipment and more. Its purpose is to control and manage the functionality of the device or system it's embedded in. Examples include software found inside routers, operating systems in smartphones and control software in automotive systems.

Embedded applications are used in a wide range of devices and systems, including:

- consumer electronics such as smartphones, smart TVs and home appliances
- automotive systems such as infotainment systems and engine control units
- industrial machinery and equipment
- medical devices

- aerospace and defence systems
- IoT devices such as smart sensors and wearable devices.

Client requirements that affect the selection of an appropriate application software category

To select the most appropriate application software category, understanding and aligning with client requirements is essential. Software type should meet their needs while considering factors such as functionality, budget, security, scalability, support and user experience.

Figure 1.5: An example of a smart home device used to control the thermostat

Table 1.7: Advantages and disadvantages of different types of application software

Application software	Advantages	Disadvantages
Closed	• Security: may be more secure as the source code is not publicly available, making it harder for people to find vulnerabilities • Control: developers have full control over the software, allowing them to maintain quality and consistency.	• Lack of customisation: users have limited ability to customise or modify the software to suit their specific needs • Dependence: users are dependent on the developer for updates, bug fixes and support • Cost: closed-source software can be more expensive which can be a barrier for some users.
Shareware	• Evaluation: users can often try the software before purchasing, allowing them to evaluate its suitability. • Flexibility: developers can offer different pricing models, such as one-time purchases or subscriptions, to meet different user needs.	• Piracy: shareware is prone to piracy, as users can easily distribute unauthorised copies. • Limited functionality: shareware versions often come with limitations or restricted features compared to the full version, which may frustrate users.
Freeware	• Cost: freeware is free to use, making it accessible to a wide range of users. • Community support: freeware often benefits from community contributions, such as bug fixes and feature enhancements.	• Sustainability: it can be challenging for developers to sustain development and support without a direct revenue stream. • Quality: without financial incentives, developers may prioritise other projects or neglect maintenance and updates.
Embedded	• Efficiency: embedded applications are designed to run on specific hardware, resulting in optimised performance. • Reliability: embedded applications are often highly reliable, as they undergo rigorous testing and optimisation for their intended use cases.	• Limited flexibility: embedded applications designed for a specific hardware may not be adaptable for other environments. • Development complexity: developing embedded systems requires specialised knowledge of the hardware and programming skills that can increase the project cost.

Functionality – If the client requires extensive customisation and specific features tailored to their unique needs, they may prefer closed software where they can work closely with developers to implement those features. For clients who need a wide range of features but have limited budget constraints, shareware or freeware options may be suitable as they often provide a broad set of functionalities at a lower cost.

Budget constraints – Clients with limited budgets may prefer freeware or open-source software as they are usually available at no cost, reducing the cost. Alternatively, clients may choose shareware software if they are willing to pay for additional features or support, but still require cost-effective solutions.

Security and compliance requirements – Clients operating in industries with strict security and compliance regulations may prioritise closed-source software due to its enhanced security features and support. Embedded applications may also be considered for industries where security and reliability are paramount, as they are often purpose-built for specific hardware.

Scalability – Clients with scalability requirements may select closed or shareware software that offers extensive customisation options and scalability features to allow for future growth.

Support and maintenance – Clients with limited IT resources may prioritise software categories that offer comprehensive support and maintenance packages. Closed and shareware software often provide dedicated support channels and regular updates. Freeware and open-source software may still be viable options if the client is comfortable relying on community support or has in-house technical expertise.

User experience and training – Clients concerned about user experience and ease of adoption may prioritise closed or shareware software with intuitive interfaces and extensive user documentation. Freeware and open-source software may require more extensive training and support to maximize user proficiency, depending on their complexity.

1.3.3 Application software types

Off-the-shelf

Off-the-shelf software is prepackaged software that is developed and sold to a wide audience. It is ready-made and typically comes with standard features and functionalities. The purpose of off-the-shelf software is to provide solutions to common problems or meet needs in a general way across various industries. It's readily available for purchase or licencing and can be deployed relatively quickly. Examples include Microsoft Office, Adobe Photoshop and QuickBooks.

Table 1.8: Advantages and disadvantages of off-the-shelf application software

Advantages	Disadvantages
• Cost-effective: off-the-shelf software is typically cheaper than bespoke solutions as development costs are spread across a wide user base. • Fast deployment: because off-the-shelf software is pre-built, it can be deployed quickly, saving time and effort. • Established support: off-the-shelf software often comes with dedicated support from the vendor, including updates, patches and user documentation.	• Limited customisation: off-the-shelf software may not fully meet the unique requirements of every organisation. • Feature overload: some off-the-shelf software may come with features that are not relevant to a particular organisation, potentially causing confusion. • Dependency on vendor: organisations rely on the vendor for updates, support and maintenance, which may lead to issues if the vendor discontinues the product or provides inadequate support.

Custom off-the-shelf

Custom off-the-shelf software is a modified version of off-the-shelf software that has been tailored to meet the specific requirements of a particular organisation or industry. The purpose of custom off-the-shelf software is to combine the advantages of off-the-shelf solutions with the ability to address specific business needs. Organisations may customise the software to integrate with existing systems or comply with requirements of their business.

Table 1.9: Advantages and disadvantages of custom off-the-shelf application software

Advantages	Disadvantages
• Tailored solutions: custom off-the-shelf software offers a balance between standard features and customisation. • Reduced development time: typically requires less time and effort compared to building a bespoke solution from scratch, leading to faster deployment. • Lower cost: customising existing software is often more cost-effective than developing bespoke solutions since the foundation already exists, reducing development expenses.	• Limited flexibility: despite customisation options, organisations may still encounter limitations in functionality or integration capabilities. • Dependency on vendor: organisations may face challenges if the vendor does not support extensive customisation or if future updates conflict with customisations made. • Potential for over-customisation: organisations may over-customise off-the-shelf software, leading to complexity, maintenance issues and challenges in future upgrades.

Bespoke

Bespoke software is developed from scratch to meet the specific requirements of a single client or organisation. It is built to meet the client's specifications and needs. The purpose of bespoke software is to provide a highly customised solution that precisely meets the client's unique business processes and objectives. It offers complete control over features, design and functionality. Bespoke software is often used when off-the-shelf solutions cannot meet the organisation's requirements, or they want to stand out from their competition.

Table 1.10: Advantages and disadvantages of bespoke application software

Advantages	Disadvantages
• Complete customisation: bespoke software offers total flexibility and customisation, allowing organisations to tailor the solution to their unique requirements. • Competitive advantage: bespoke software can provide a competitive edge by addressing specific business needs and improving efficiency. • Ownership and control: organisations have full ownership and control over bespoke software, including intellectual property rights, allowing them to make changes and enhancements as needed.	• Higher cost: bespoke software development is typically more expensive than off-the-shelf solutions. • Longer development time: building bespoke software from scratch requires more time and resources compared to deploying off-the-shelf or customised solutions. • Higher risk: custom software development carries risks, including losing site of the aims, technical challenges and project delays, which may impact timelines and budgets.

Client requirements that affect the selection of an appropriate application software type

To find the best fit of application software type to meet the client requirements, the following factors may need to be considered:

Client specific requirements – If the client's requirements are very specific or complicated, they may need a bespoke solution. This would allow for complete customisation. However, if the requirements are more common, there may be an off-the-shelf product that will meet their needs.

Budget and time constraints – Clients with a limited budget or short time frame available may not be able to afford a custom off-the-shelf or bespoke solution. An off-the-shelf solution is cheaper and more quickly deployed, however, it would still need to be a good fit for the client's needs to be a suitable option.

Need for flexibility – Requirements can change over time, for example as the organisation grows. If this is the case then a solution that allows for flexibility, such as a custom off-the-shelf solution may offer this.

Industry-specific requirements – Clients may have industry specific needs or regulatory requirements to meet. In such cases, the client may need software that is tailored to address these requirements that could be best met by bespoke software.

Integration with existing systems – Clients with existing systems may want software solutions that seamlessly integrate with their current setup. This compatibility is best achieved through a custom off-the-shelf or bespoke solution.

User experience – Client requirements related to user experience, interface design and usability can influence the choice of application software type. Off-the-shelf solutions often come with existing user interfaces and functionalities which may or may not be suitable. Bespoke solutions, however, can be tailored to provide an intuitive user experience aligned with the client's preferences and users.

Support and maintenance – Ongoing support, maintenance and updates are all important when selecting a software type. Off-the-shelf solutions typically come with vendor support and regular updates, while bespoke solutions may require dedicated maintenance resources or ongoing support contracts to provide a good experience for the end user.

Quick check 3

1 Summarise the differences between off-the-shelf, custom off-the-shelf and bespoke software application types.

2 Consider the factors affecting choice of application software type. If you were going to put them in order from the most to least important, which would be first and why?

Practice questions 1

ABC Health and Fitness is a leading wellness organisation committed to promoting healthy lifestyles through nutrition and exercise. They are seeking to develop a new mobile application called HealthPro that aims to simplify meal planning, promote balanced nutrition and empower users to make healthier food choices.

1 Explain the relationship between a program and an application. **[2]**

2 HealthPro will be designed for a proprietary operating system. Describe **one** advantage of this choice. **[1]**

3 Identify **one** type of application for HealthPro. **[1]**

4 Identify **two** possible characteristics of the application type chosen in **question 3**. **[2]**

TA2 Software development models

Learning intentions

This topic is about the different software models that may be used to create an application and their common phases.

It covers:

2.1 Software development models

2.2 The common phases of software development models

2.1 Software development models

Software development models are structured approaches to design used by software development teams to structure, plan and control the process of creating software. The models provide a framework that outlines the sequence of activities, tasks and deliverables involved in the software development lifecycle.

Software development models are used for the following reasons:

- They provide structure, outlining the activities from start to finish.
- They improve communication by making sure everyone understands their tasks and how they contribute to the final product.
- They help manage risk by identifying issues at an earlier stage.
- They support best practice and meeting quality standards.

Some common models used are highlighted in Table 1.11.

Table 1.11: Types of software development model

Traditional model	Prototyping model	Iterative model
Waterfall	• Rapid throwaway • Incremental • Evolutionary.	• Rapid Application Development (RAD) • Spiral • Agile.

Traditional model

Waterfall

The **waterfall model** is a linear and sequential approach to software development. It consists of distinct phases, such as requirements gathering, design, implementation, testing, deployment and maintenance. Each phase must be completed before moving on to the next, resembling a waterfall flowing downwards.

This model is well-suited for projects with clear and stable requirements or projects with strict regulatory or compliance requirements where documentation is crucial.

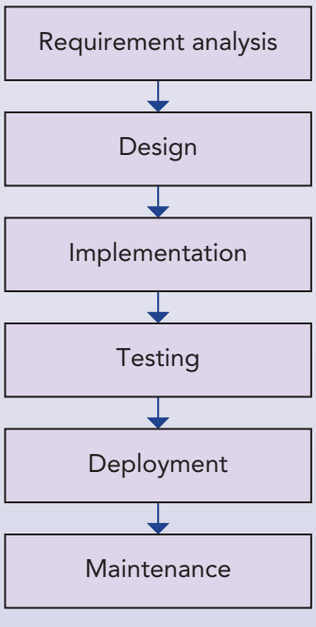


Figure 1.6: Stages of a waterfall approach to software deployment

Table 1.12: Advantages and disadvantages of a waterfall software development model

Advantages	Disadvantages
• Clear and well-structured process • Easy to understand and manage • Each phase has defined deliverables, making progress measurable.	• Lack of flexibility to accommodate changes in requirements • Limited opportunities for customer feedback and validation during development.

Prototyping model

Rapid throwaway

The rapid throwaway model focuses on quickly building prototypes or mock-ups of the software to gather feedback and check requirements. **Prototype models** are often incomplete or of low quality, intended for demonstration and discussion purposes. Once feedback is received, the prototype is discarded and development begins using a different model or approach.

This approach is used for projects where the primary goal is to explore different design options or concepts quickly. It is also used for projects with high uncertainty or risk where early experimentation is necessary before committing to a final design.

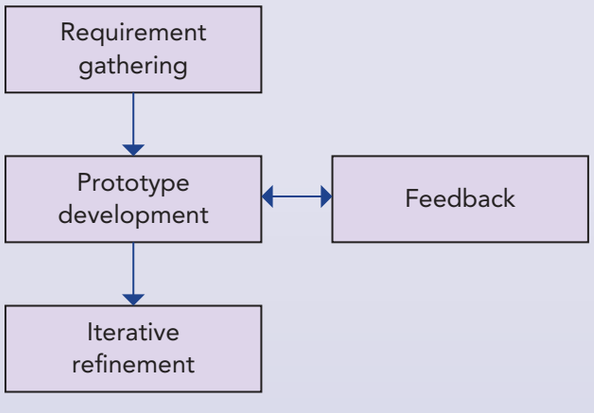


Figure 1.7: Simple diagram showing the rapid throwaway model

Table 1.13: Advantages and disadvantages of a rapid throwaway software development model

Advantages	Disadvantages
• Increased speed of version development • Evolve ideas quicker • Lower labour costs/ staffing • Faster decision making • Can show client versions easier/ quicker • Reduced documentation.	• Can develop too many versions that are not needed • Increased costs by developing many versions • Confusion on feedback when many versions are created quickly • Lack of analysis at each stage of prototype development • Final product requires complete redevelopment from scratch after prototyping phase • May lead to wasted resources if prototyping reveals solution is not suitable.

Incremental

The incremental software development model involves breaking down the project into smaller, manageable modules or increments. Each increment represents a complete subset of functionality that can be developed, tested and delivered independently. Development occurs with each iteration or cycle adding new features or enhancements to the software. Stakeholders can review and provide feedback on each increment, allowing for continuous improvement and refinement.

This approach is suited to projects where requirements can be divided into distinct, prioritised increments. They might also be long-term projects with evolving requirements in which delivering early and frequently is important.

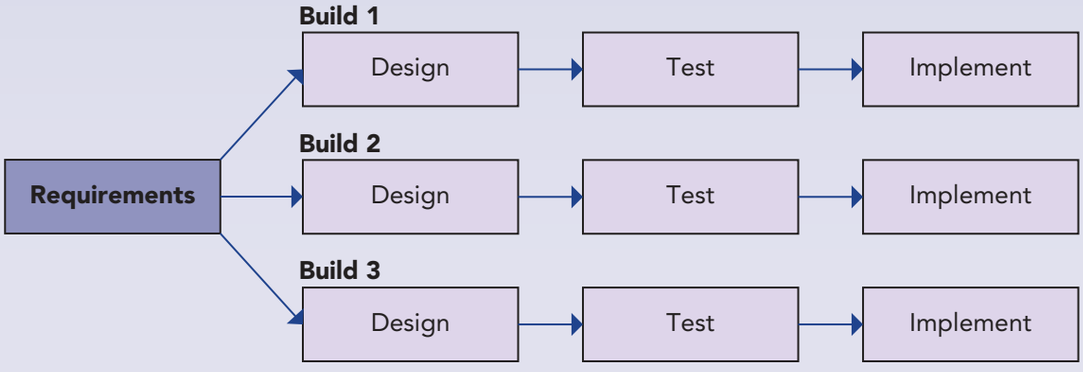


Figure 1.8: Diagram showing how each increment builds on work completed in previous iterations in the incremental model

Table 1.14: Advantages and disadvantages of an incremental software development model

Advantages	Disadvantages
• Flexibility to accommodate changes and adapt to evolving requirements • Opportunities for feedback at each iteration.	• Requires careful planning to define increments • Risk of scope growing too much if requirements are not managed effectively.

Evolutionary

The evolutionary model focuses on **iterative** development and refinement of the software through multiple cycles or iterations. Each iteration involves gathering requirements, designing, implementing, testing and evaluating the software. It allows for progressive development of requirements and design, with feedback from stakeholders incorporated into subsequent iterations.

This model is suitable for projects with evolving or unclear requirements where exploration and experimentation may be beneficial. They may also be used for technically complex projects or those where the solution is not clearly understood at the start.

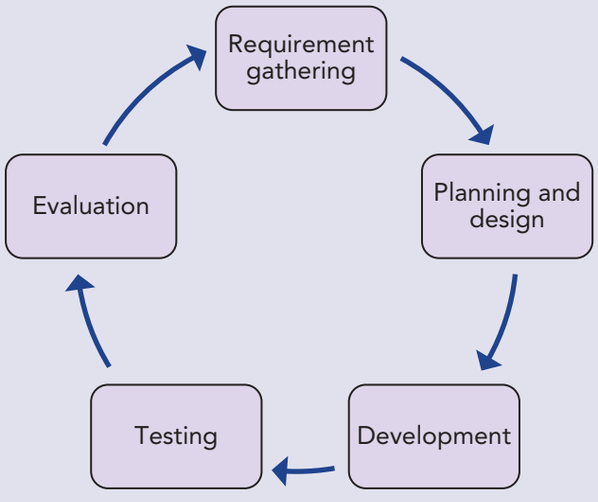


Figure 1.9: In evolutionary software development, feedback from stakeholders on one cycle is incorporated into the next cycle

Table 1.15: Advantages and disadvantages of an evolutionary software development model

Advantages	Disadvantages
• Iterative and incremental development allows for continuous improvement and refinement • Provides flexibility to respond to changing requirements and market conditions.	• Requires active user involvement and frequent feedback • May lead to higher initial development costs.

Iterative model

Rapid Application Development (RAD)

The RAD model is a type of incremental model that emphasises rapid prototyping, iterative development and user involvement. It involves using pre-built components, reusable code and prototyping tools to accelerate development. It encourages collaborative and interactive development, with close involvement of end users and stakeholders.

This model is used for projects with tight deadlines or those the need to gather quick feedback from stakeholders or end users. They are typically used for small to medium-sized projects where speed and flexibility are more important than extensive planning and documentation.

Table 1.16: Advantage and disadvantage of a RAD software development model

Advantage	Disadvantage
Accelerated development through use of pre-built components and rapid prototyping.	Requires a skilled and experienced team.

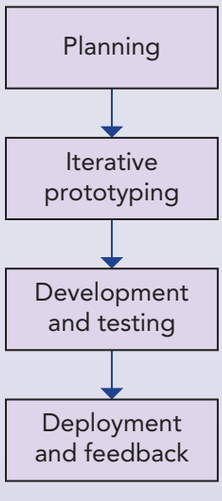


Figure 1.10: In the RAD model, development cycles are short and focused on delivering working prototypes quickly

Spiral

The spiral model combines elements of both iterative and waterfall approaches, emphasising risk management and incremental development. It consists of multiple cycles or spirals, each of which includes phases of planning, risk analysis, engineering and evaluation. The spiral model allows for progressive development of requirements and early identification of potential risks.

This model is used for projects with significant technical risks or uncertainties that need to be managed throughout the process. These may be large-scale projects where managing risk is important for success.

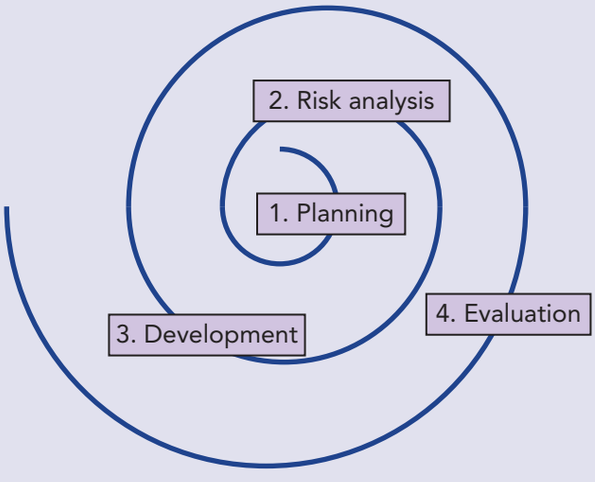


Figure 1.11: Phases within a spiral model

Table 1.17: Advantages and disadvantages of a spiral software development model

Advantages	Disadvantages
• Enables early risk identification and mitigation • Flexibility to accommodate changes and adjustments throughout the development process.	• Can be complex and resource-intensive to manage, particularly for small projects • Requires skilled and experienced team to effectively manage risk analysis.

Agile

Agile methodologies, such as Scrum and Kanban, prioritise flexibility, collaboration and iterative development. Scrum is primarily used for software development projects. Kanban, on the other hand, is more visual, using a board to represent work items moving through various stages of a process.

Agile approaches involve breaking the project into smaller iterations or sprints, typically lasting one to four weeks. Development teams work closely with stakeholders, adapt to changing requirements and deliver working software incrementally.

This model is used for projects with rapidly changing or evolving requirements where stakeholder collaboration and customer feedback are essential. It is typically used for small to medium-sized projects.

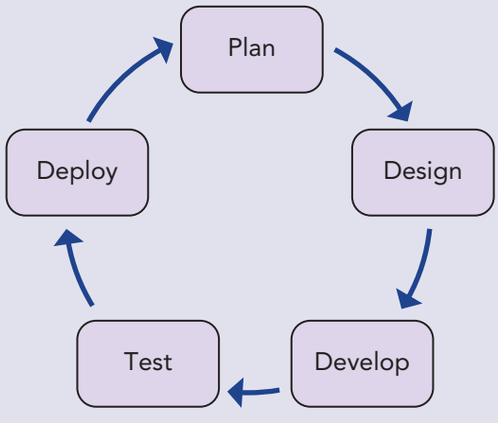


Figure 1.12: In agile software development, activities are divided into small, manageable iterations

Table 1.18: Advantages and disadvantages of an agile software development model

Advantages	Disadvantages
• Emphasis on customer collaboration ensures software meets user needs and expectations • Continuous improvement and adaptation to evolving requirements.	• Requires highly collaborative team with strong communication skills • May not be suitable for projects with rigid or fixed requirements.

Using software development models offers several advantages. For example:

- Structured approach: development models provide a structured framework for organising and managing the software development process. This structured approach helps teams to plan, execute and monitor the project effectively.
- Clarity and communication: models facilitate communication and collaboration among project stakeholders by establishing a common understanding of the development process, milestones and deliverables. Clear communication helps in aligning expectations and resolving conflicts efficiently.
- Quality checking: development models emphasise quality throughout the software development lifecycle. By including testing activities at various stages, models help in identifying defects early, ensuring that the final product meets the required quality standards.

While software development models can have many advantages, there can also be many disadvantages. Some of these include:

- Rigidity: some development models, such as the waterfall model, can be rigid and inflexible, particularly when it comes to accommodating changes in requirements or responding to unexpected challenges. This rigidity may result in delays or cost overruns if not managed properly.
- Level of administration: certain models, especially those with extensive documentation and formal processes, can introduce administrative burden on the development team. This additional overhead may slow down the development process and increase project costs.

- Limited flexibility: some models, such as waterfall, follow a sequential and linear approach, which may not be well-suited for projects with evolving or unclear requirements. Such models may lack the flexibility to adapt to changing customer needs or market conditions.

These limitations will need to be carefully considered and addressed to ensure successful project outcomes.

Learning in context 2

Selecting a software model type

XYZ Tec, a leading technology company, wants to develop a new mobile application to enhance customer engagement and expand its market reach. Facing tight timelines and evolving requirements, XYZ Tec needed to carefully choose a project management model that would ensure efficient development, adaptability to changes, and timely delivery of the app.

At the start of the project, XYZ Tec chose to follow the waterfall project management model.

1 Clarity of requirements: The initial stage of the project had well-defined requirements, making it suitable for the waterfall model's linear and sequential movement through stages such as requirements gathering, design, development, testing and deployment.

2 Predictability: By following a defined plan and sequence of activities, XYZ Tec felt there would be greater predictability in project timelines and deliverables, crucial for meeting stakeholder expectations and business objectives.

3 Documentation: The waterfall model emphasises comprehensive documentation at each stage, providing clear guidelines and reference points for development teams and stakeholders, so reducing the chance of misunderstandings.

However, as the project progressed, XYZ Tec faced evolving customer needs and market changes. They felt a shift to a more adaptive and iterative project management type was required. Consequently, XYZ Tech transitioned to the agile project management model.

They made the change for the following reasons:

1 Flexibility: Agile's iterative nature allowed XYZ Tec to respond quickly to changing requirements and incorporate feedback from stakeholders, ensuring the app remained in line with customer expectations and market trends.

2 Continuous improvement: Agile promotes continuous iteration and improvement, enabling XYZ Tec to prioritise features and deliver incremental releases that provide real benefits to users, creating a culture of innovation and responsiveness.

3 Enhanced collaboration: Agile fosters close collaboration among teams, enabling communication, knowledge sharing and collective problem-solving, leading to higher-quality product and stronger team morale.

By initially adopting the waterfall model and later changing to the agile model, XYZ Tec balanced the need for structured planning with the flexibility to adapt to changing requirements and the market.

Quick check 4

1 Imagine that you are developing a new mobile app for your school or college. How would you decide which software development model to use for the project? Make a list of factors and briefly explain why they are important.

2 Create a mind map to draw out the key features of each of the following types of software development model: traditional model, prototyping model, iterative model.

2.2 The common phases of software development models

Planning

The planning phase of the software development model lays the foundation for a successful project by defining project goals, scope, resources, schedules and risk management strategies. The specifics of the planning phase will vary depending on the model chosen but will usually examine the project requirements and the **feasibility** of the project.

Requirements

In software development, the requirements provide information on what the software should do and how it should work. They are a detailed list of everything the software needs to do to be successful.

This involves gathering and analysing requirements from stakeholders, including clients, end users and other relevant parties.

The first stage is to identify stakeholders, those who have an interest in the software being developed. In addition to the client, this might include subject matter experts and regulatory bodies. To understand the requirements, you could use a range of methods such as:

- Interviews: in-person or online meetings with a series of open-ended questions to establish stakeholder perspectives and requirements
- Workshops and focus groups: in-person or online meetings with a group of people to hold collaborative discussions, encouraging active participation and engagement from a range of stakeholders
- Surveys and questionnaires: used to gather quantitative data and feedback on specific aspects of the software requirements
- Prototyping and mock-ups: develop prototypes, mock-ups or proof-of-concept demos to visualise and validate requirements with stakeholders.

As part of this phase, iteration may occur as stakeholders provide feedback or as the team gains a deeper understanding of the project scope.

Feasibility

In analysing the feasibility of a software development project, it is important to consider several factors.

- Technical feasibility: this examines whether the project is possible using the available technology, tools and resources. This includes a consideration of system functionalities and the expertise of the people available to take part in the project.
- Economic and/or commercial feasibility: this will look at whether the project would work financially when the costs of development are considered when weighed up against potential benefits. Potential benefits might include increased revenue, cost savings, improved efficiency or competitive advantages.
- Schedule feasibility: this evaluates whether the proposed project can be completed within the desired timeframe and deadlines. It involves examining the time required for each element

of the project and the availability of staff and resources. Project scheduling tools such as Gantt charts may be used.

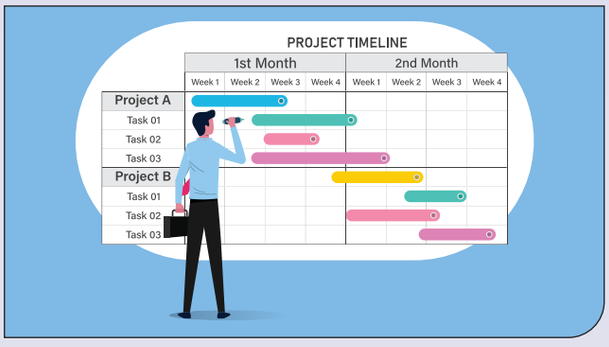


Figure 1.13: A Gantt chart is a visual tool to show tasks, their duration and their dependencies

Design

The design phase in software development is the stage where the requirements gathered during the planning phase are translated into a detailed blueprint or design for the software system. It involves creating a comprehensive plan that outlines how the software will be structured, organised and implemented to meet the specified requirements.

Key activities involved in the design phase include Interface Design, Architectural Design and Detailed Design.

Interface design

Interface design focuses on designing the User Interface (UI) and User Experience (UX) aspects of the software. It involves creating layouts, navigation flows and interaction patterns that are intuitive, visually appealing and user-friendly.

Interface design considers factors such as usability, accessibility and responsiveness to ensure that the software is easy to use and accessible to a diverse range of users.

Common tasks in interface design include:

- Wireframing: creating simplified sketches or mock-ups of the UI to visualise the layout and structure of screens or pages.
- Prototyping: building interactive prototypes or mock-ups to simulate the behaviour and functionality of the UI.
- User testing: conducting testing with real users to gather feedback and refine the UI design based on their preferences and needs. The aim of this is to uncover issues with user interaction with the software, understand user behaviour and get feedback to improve the overall user experience.
- Graphic design: designing visual elements such as icons, images, colours and typography to enhance the aesthetic appeal and branding of the software.

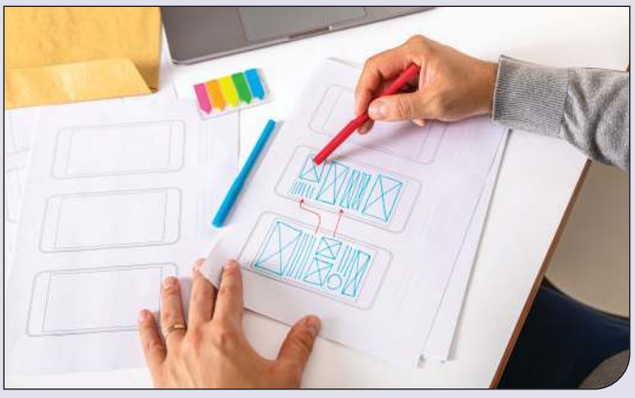


Figure 1.14: Wireframes can be used to plan the layout of screens on mobile applications

During this phase, iteration may occur as initial designs are reviewed, refined and adjusted based on feedback or changing requirements.

Architectural design

Architectural design focuses on defining the overall structure and organisation of the software system. It identifies the main parts of the system and how they work together. It is focused on making sure the system can grow, be fixed easily, work fast and stay safe.

Common tasks in architectural design include:

- System decomposition: breaking down the system into smaller, manageable components or modules that can be developed and maintained independently

- Design patterns: developing common solutions to recurring problems that are encountered when building programs
- Dependency management: managing dependencies between components and modules to ensure they are not too reliant on one another
- Trade-off analysis: checking that design choices match what the project needs and can handle.

Detailed design

Detailed design is like drawing a detailed map for building software. It's about figuring out how each part of the software will work inside and what it needs to do its job. This includes designing how the different pieces will talk to each other and making sure they're set up correctly.

Some common tasks in detailed design are:

- Component design: defining the structure and connections of each part of the software
- Algorithm design: creating step-by-step instructions for the software to solve specific problems
- Data design: deciding how information will be organised and stored
- Error handling: planning how the software will handle mistakes or problems to make sure it keeps running smoothly.

Constructing/creation

The construction phase of a software development project focuses on turning design concepts into a working software solution. This includes coding, testing, debugging, documentation and version control, so users can go back to previous versions if needed. This is a crucial stage where the software product begins to take shape.

Tasks involved in the construction phase include coding, testing and debugging.

- Coding: developers write the code according to the specifications outlined in the design documents. This involves translating design concepts into programming languages and creating algorithms to build the software solution.
- Testing: as code is written in the construction/creation phase, developers perform tests to ensure that the software behaves as expected. Testing helps identify and fix defects early in the development process, improving code quality and reliability.
- Debugging: developers troubleshoot and debug issues identified during testing. This involves identifying the causes of problems, fixing errors in the code and ensuring that the software behaves as expected under various conditions.

Testing

In addition to testing during the construction phase, the software will also go through a thorough evaluation to ensure it meets quality standards, functions correctly and meets user requirements.

During this phase, functional testing will take place. This involves checking that each function or feature of the software performs as it is supposed to according to specified requirements. It might also include regression testing. This is the process of retesting the software after code changes or modifications have been made to make sure that existing functionalities are not negatively affected.

Examples of tasks that may be included in the testing phase include:

- Unit testing: testing individual parts of the software to ensure they work correctly
- Integration testing: checking that different parts of the software work together as expected
- Compatibility testing: ensuring the software works correctly across different devices, browsers, and operating systems
- Usability testing: evaluating the user interface and user experience to make sure it is intuitive and user-friendly.

Implementation

Once the system has been thoroughly tested, it is ready for implementation.

The implementation phase is when the software is deployed and integrated into an organisation's existing systems and processes. This phase may include the installation, configuration and testing to ensure it works as expected. It may also include training users on how the software works and providing initial support to ensure the software features are being used successfully.

There are three different approaches to implementation: phased, parallel and big bang implementation, each with its own advantages and disadvantages.

Phased

Phased implementation involves rolling out the new software or changes gradually, in phases or stages. There might be certain features or functionalities that will be delivered in chunks over time. This reduces risk as testing can be carried out at each stage and eases the transition for users by allowing them to adapt gradually to the new system. However, this will prolong the overall implementation timeline as each phase requires time and resources and could cause frustration for the end user.

Parallel

Parallel implementation involves running both the old and new systems alongside one another for a period of time. This provides a safety net by allowing users to go back to the old system if issues arise with the new system. It also means that users can gradually transition to the new system at their own pace, which reduces disruption. However, organisations have the cost of two systems to maintain and support, which can be expensive. It can also be confusing for end users to have two systems at the same time.

Big bang (crash)

Big bang implementation involves delivering the new system or changes all at once, replacing the old system entirely in a single operation. This approach offers the fastest implementation timeline as the new system is deployed in one step. It also simplifies management and coordination as the focus is on a single switchover event. However, it is high risk as, if issues arise with the new system, there is no fallback option. It also forces users to adapt quickly to the new system without a transition period.

During this phase, iteration may occur as issues are identified and addressed, or as user feedback leads to updates or enhancements for the software.

Common tasks during the implementation phase include:

- Implementation plan: outline the steps, timeline and resources needed for the implementation phase
- Training: organise and deliver training to ensure users are comfortable and able to use the new system
- User support: offer helpdesk services and troubleshooting to aid users during the transition period.

Documentation creation

To support implementation of the software, user documentation needs to be created to guide end users through the software interface, features and functionalities. Clear, well written documentation is key to improving the user experience and supporting the successful adoption of the software.

Tasks might include:

- **Audience:** understanding the needs, skill level and preferences of the target audience
- **Planning:** deciding the scope of the content of the documentation by consulting with the target audience
- **Writing content:** writing clear and concise content, avoiding technical terms and jargon.
- **Visual aids:** creating visual aids such as screenshots, diagrams and videos to make the documentation more engaging and accessible.

Maintenance

Once the software has been implemented, maintenance is important to ensure the ongoing functionality, stability and usability of the software. This is essential for ensuring the long-term success of the project.

Here are the main tasks involved in the maintenance phase:

- **Bug fixing:** identifying, prioritising and fixing any bugs, errors or issues that occur in the software. This may involve troubleshooting reported problems, analysing causes and resolving the issues.
- **Enhancements:** adding new features, functionalities or improvements based on user feedback, changing requirements or advances in technology.
- **Performance optimisation:** identifying and addressing performance issues that affect its speed, responsiveness or scalability, for example.
- **Security updates:** monitoring and addressing security vulnerabilities, threats and risks to ensure the software remains secure and protected against potential cyberattacks or data breaches.
- **Compatibility maintenance:** ensure that the software remains compatible with changing platforms, operating systems, browsers and third-party integrations.

During this phase, iteration may occur as new features are added, existing features are improved or as bugs are reported and resolved.

Quick check 5

1 Create an infographic that summarises each of the phases of software development and highlights **at least two** tasks that might feature in each phase.

2 Explain to someone the differences between the three approaches to implementation.

Practice questions 2

Nimble Motors is a leading car dealership and service centre dedicated to providing high-quality vehicles and exceptional customer service. They want to update their existing mobile application to allow them to showcase their latest vehicle stock and enable customers to conveniently book their cars for repair and maintenance services. Their goal is to enhance the customer experience, streamline their operations and strengthen their online presence.

1 Discuss whether Nimble Motors should use the waterfall model for the development of the application. **[9]**

In your answers, you **must** write about:

- any advantages to Nimble Motors
- any disadvantages to Nimble Motors
- whether you would recommend that Nimble Motors use the waterfall model and your reasons.

2 Nimble Motors have decided on a big bang implementation approach.

Describe what is meant by big bang implementation. **[2]**

TA3 Planning application development projects

Learning intentions

This topic is about why planning a project is important, planning considerations and the impact of planning on application development. It also explores the component parts of a variety of planning tools that might be used, their advantages and disadvantages and how they can be used effectively.

It covers:

3.1 Planning projects

3.2 Project planning tools

3.1 Planning projects

Purpose of planning projects

Project planning is crucial to the success of an application development project for several reasons. First, it provides a clear vision and **objectives** for the project. This means that all stakeholders have an understanding about what needs to be done. A project plan will also set out the timeline for the project and **milestones** that need to be met along the way. This helps track progress and make sure that the project is completed on time.

A project plan will also identify and allocate **resources** such as time for different tasks and activities, **budget** and **personnel** needed. This means that resources can be used as efficiently as possible, and wastage reduced. This also helps to accurately control costs and manage expenses across the project lifecycle.

Planning also helps in identifying potential risks and developing strategies to reduce or remove them. By foreseeing and tackling risks early on, the project stands a better chance of success. Planning also helps with quality assurance by defining quality standards and procedures. This ensures that the final product meets the desired quality criteria and satisfies user requirements.

Finally, a project plan facilitates better communication and collaboration among team members and stakeholders. It ensures that everyone involved in the project understands their roles and responsibilities and those of others within the project.

Impact of not planning a project

Overall, it is preferable to plan carefully for a project for it to be successful. If a project is not adequately planned, it can result in the partial or total failure of a project to deliver a suitable final product.

For example, if project objectives are vague or poorly defined this can lead to confusion among team members and stakeholders. This might result in delays to the project or an unsuitable final product. If a project timeline is not established, it's easy for projects to fall behind schedule, leading to missed deadlines and potential business impacts such as increased costs or loss of revenue.

Resource management is key to project success. Where this is not considered, there may be an inefficient allocation of resources, leading to delays and poor-quality outcomes. This can also lead to unexpected expenses, resulting in budget overruns and financial strain on the organisation. Additionally, where risks are not considered at the outset of a project, it becomes more vulnerable to unexpected challenges, which can derail progress and impact success.

Finally, without adequate project planning, communication among team members and stakeholders can suffer. This can lead to misunderstandings and conflict about the project and its direction.

Table 1.19: Advantages and disadvantages of planning application development projects

Advantages	Disadvantages
• Clarity of objectives: planning helps to define clear project objectives and scope, ensuring that all stakeholders understand what needs to be achieved. • Resource optimisation: proper planning enables efficient allocation and use of resources such as time, budget and manpower, reducing wastage and maximising productivity. • Risk mitigation: planning involves identifying and addressing potential risks early in the project lifecycle, reducing the likelihood of costly setbacks and delays. • Timeline management: planning allows for the setting of realistic timelines and milestones, enabling better tracking of progress and ensuring timely delivery of the project. • Quality assurance: planning includes defining quality standards and processes, ensuring that the final product meets the desired level of quality and user requirements. • Cost control: by estimating costs accurately and controlling expenses throughout the project, planning helps prevent budget overruns and financial issues.	• Time consuming: planning can be time-consuming, especially for complex projects, which may delay the start of a project. • Overplanning: excessive planning, where too much time is spent planning and not enough on actual execution, can result in project delays. • Inflexibility: overly rigid plans may not be flexible enough to reflect changes or unforeseen circumstances, leading to difficulties in adapting to evolving project requirements. • Costly changes: if planning is not done thoroughly, it may lead to costly changes later in the project lifecycle, increasing expenses and delaying delivery. • Dependency on assumptions: plans are often based on assumptions about future events or conditions, which may turn out to be incorrect, leading to deviations from the planned course of action.

Planning considerations

To successfully plan a project, the following areas must be considered:

- budget
- constraints
- legislation
- resources
- success criteria
- time.

Budget

The budget is the amount of money that is available for the project. The project plan should ensure that the overall development budget is not exceeded. The budget directly affects the scope and scale of the application development. It influences the size of the development team, the technology that can be used and the level of customisation or features that can be implemented. A limited budget may require prioritising certain features over others or seeking cost-effective solutions.

Constraints

Constraints are limitations or boundaries that may affect the project and could be technological, environmental, regulatory or organisational constraints. Within the project plan it is important to understand and address constraints to ensure project success. Constraints influence decision-making throughout the development process. Technological constraints may dictate the choice of platforms or tools, while regulatory constraints such as data protection laws may affect how user data is handled. Ignoring constraints can lead to compliance issues, project delays or unexpected costs.

Legislation

Legislation refers to laws or a set of laws that have been passed by a government. Within the planning phase of the project, it is vital relevant legal and ethical issues are considered.

The following areas of legislation may be relevant to application software development:

- Copyright
- Data protection
- Electronic communications.

Copyright law protects original works of authorship, including software code, images, videos and sound. It ensures that developers have exclusive rights to their code and prevents unauthorised copying, distribution or modification of their work. Developers need to be mindful of copyright law when creating software to avoid infringing on others' intellectual property rights. Failure to comply with copyright law can lead to legal disputes, fines and damage to reputation.

Data protection legislation aims to protect individuals' privacy rights and regulate the processing of personal data. In software development, compliance with data protection laws is crucial when handling personal user information such as name, email address and location data, for example. Software developers must use appropriate data protection measures such as encryption and CAPTCHA to ensure the confidentiality, integrity and availability of user data.

Different countries have different data protection laws and regulations. For example, the European Union's General Data Protection Regulation (GDPR) has strict requirements for data protection, while other countries may have less strict rules. Organisations must follow the data protection laws of all countries involved in the data transfer. This can be made more complicated when data is transferred across borders and requires a thorough understanding of international regulations.

Electronic communications are also subject to legislation, including email, messaging and electronic marketing. Software developers must comply with electronic communications legislation when developing applications that involve electronic messaging or marketing activities. This includes obtaining consent from users before sending commercial communications, providing means of opting out and ensuring compliance with rules regarding spam emails and electronic marketing.

Making sure the application complies with legislation is critical for legal and ethical reasons. Failure to stick to relevant laws and regulations can lead to financial penalties, and damage to the organisation's reputation.

Resources

Resources include the human, technological and physical assets required to complete the project successfully. These should be carefully considered as part of the project plan so there are enough skilled personnel, hardware and software, and support systems for efficient application development. Inadequate resources may lead to project delays, increased costs and impact on quality.

Success criteria

Success criteria define the measurable factors that determine project success. When planning a project these need to be identified (see TA2.2), so that progress to the desired outcome can be measured allowing those involved to track the progress and make any adjustments if needed. Success criteria guide decision-making and project priorities during application development. They help ensure that the developed application meets user needs, achieves business objectives and delivers expected benefits.

Time

Time is a critical constraint in project management. When planning, meeting deadlines and delivering the project on schedule is essential to meet stakeholder expectations and minimising additional costs. Time constraints may mean prioritising essential features to ensure deadlines are met. Failure to manage time effectively can lead to project delays, missed opportunities and dissatisfaction among stakeholders.

Quick check 6

1 Summarise the reasons why planning a project is important.

2 Create a mind map to draw out key points about factors to consider when planning a project.

3.2 Project planning tools

Each of these project management tools offers unique advantages and disadvantages, and the choice of tool(s) depends on the specific needs, complexity and goals of the project, as well as the preferences and expertise of the project team.

Often, a combination of tools may be used to address different aspects of project planning to ensure a successful outcome.

Arrow diagram

An **arrow diagram** is a graphical representation used in project planning to show the order of activities and their dependencies within a project. It consists of nodes (or circles) that represent events and arrows that represent activities. The arrows show the relationships between activities and the direction of travel. Dummy arrows can also be used to represent relationships between activities where there is no direct link. Dummy arrows are drawn as dashed lines.

Arrow diagrams are often used together with **Critical Path Method (CPM)** or **Program Evaluation and Review Technique (PERT)** to analyse project schedules, identify critical paths and work out the earliest and latest start and finish times for activities. They provide a visual representation of the project's workflow, helping project managers and teams understand the project's structure and dependencies, and helping to schedule and plan resource allocation.

Table 1.20: Advantages and disadvantages of arrow diagrams

Advantages	Disadvantages
• Provides a visual representation of the project's tasks and their dependencies • Helps in identifying the critical path and understanding the sequence of tasks • Useful for complex projects with numerous tasks.	• Can become complex and difficult to understand for large projects • May not clearly show task durations or resource allocation • Less commonly used compared to other tools.

Critical Path Analysis (CPA)/Critical Path Method (CPM)

Critical Path Analysis (CPA) or Critical Path Method (CPM) is a project management technique used to plan and manage complex projects. It helps project managers identify the longest sequence of dependent activities and work out the shortest possible time for completing a project.

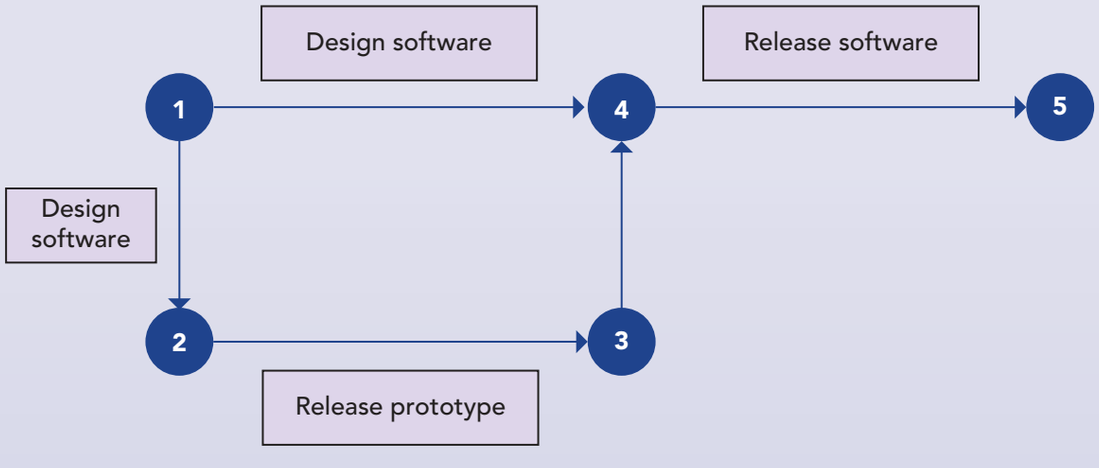


Figure 1.15: Example of an arrow diagram with event nodes and arrows to show path through a project

The key features are:

- Tasks and activities: the work required to complete the project, clearly defined, with specific start and end points.
- Dependencies: tasks often depend on each other, meaning that the start or completion of one task may affect the start or completion of another.
- Duration: the time required to complete each task. This can be estimated based on historical data, expert judgment or other estimation techniques.

A visual representation of these elements is plotted on a network diagram. It can be constructed using nodes (representing tasks) and arrows (representing dependencies). Network diagrams help in identifying the critical path and understanding the sequence of tasks. The critical path is the longest sequence of tasks through the project network, meaning that if any task on the critical path is delayed, the entire project will be delayed.

Table 1.21: Advantages and disadvantages of CPA

Advantages	Disadvantages
• Identifies the critical path, helping to prioritise tasks and focus resources on critical activities • Provides a clear understanding of project dependencies and the shortest possible duration for project completion • Helps in scheduling, resource allocation and risk management.	• Requires accurate task duration estimates, which can be challenging to establish • Does not account for uncertainties or variations in task duration • Can be complex to implement and may require specialised software.

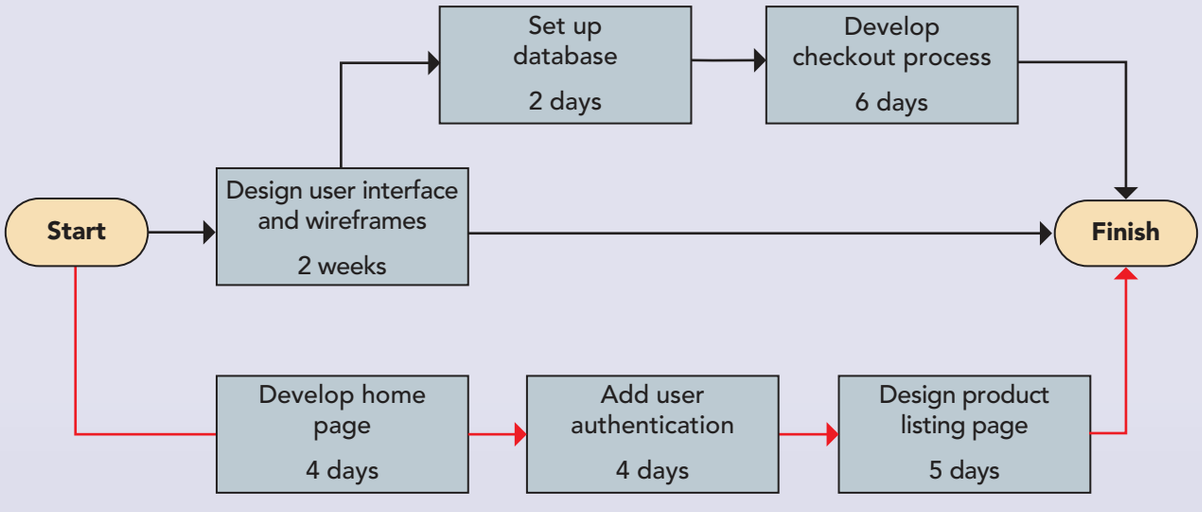


Figure 1.16: Example of a Critical Path Method (CPM) plan for an online shopping application

Flowchart

A **flowchart** can be a valuable tool for visualising the sequence of tasks, dependencies, decision points and overall workflow of a project. They are commonly used to communicate the logic and flow of the software's processes. They support the planning process by providing a clear visual roadmap of the stages in the project from the initial idea to completion.

Table 1.22: Commonly used symbols used on a flowchart when mapping out a process within a project

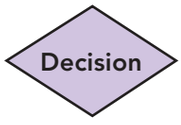
Symbol	Symbol name	Description
	Start/End	These symbols represent the beginning and end of the project flowchart.
	Process/Task	These symbols represent specific tasks or activities within the project.
	Decision	These symbols represent points in the project where a decision needs to be made, leading to different paths or outcomes.
	Data Input/Output	These symbols indicate where data is entered into or extracted from the project.
	Arrows	These connect the various symbols in the flowchart, indicating the sequence of tasks and the flow of the project.

Table 1.23: Advantages and disadvantages of flowcharts

Advantages	Disadvantages
• Offer a visual representation of project workflows, decision points and process sequences • Easy to understand and communicate, making them useful for team collaboration and stakeholder engagement.	• May become too complex for large projects • Limited in their ability to represent detailed task durations, resource allocations or dependencies • Not specifically designed for project management.

Gantt charts

Gantt charts are often found within project management software. They play an important role in project planning as they visually represent the schedule of tasks, their durations, dependencies and progress over time. This helps plan the timeframe for the project, how tasks should be ordered and how resources should be allocated across the project.

The key components and conventions include:

- Task list: a column on the left side of the Gantt chart that outlines all the tasks or activities involved in the project. Each task is typically listed as a row in the chart.
- Time scale: the horizontal axis of the Gantt chart represents time, usually broken down into days, week, or months, depending on the project's timeline. The time scale provides a reference for scheduling tasks and tracking progress over time.
- Bars/blocks: bars or blocks represent individual tasks or activities. Each task is depicted as a horizontal bar covering the duration of the task along the time axis. The length of the bar linked to the task's duration. Shading may also be used to show task completion.

- Dependencies: dependencies, shown by arrows or lines connecting tasks, indicate the sequence in which tasks must be completed.
- Milestones: significant events or achievements within the project timeline, represented on the Gantt chart as diamond-shaped symbols placed at specific points along the time axis.

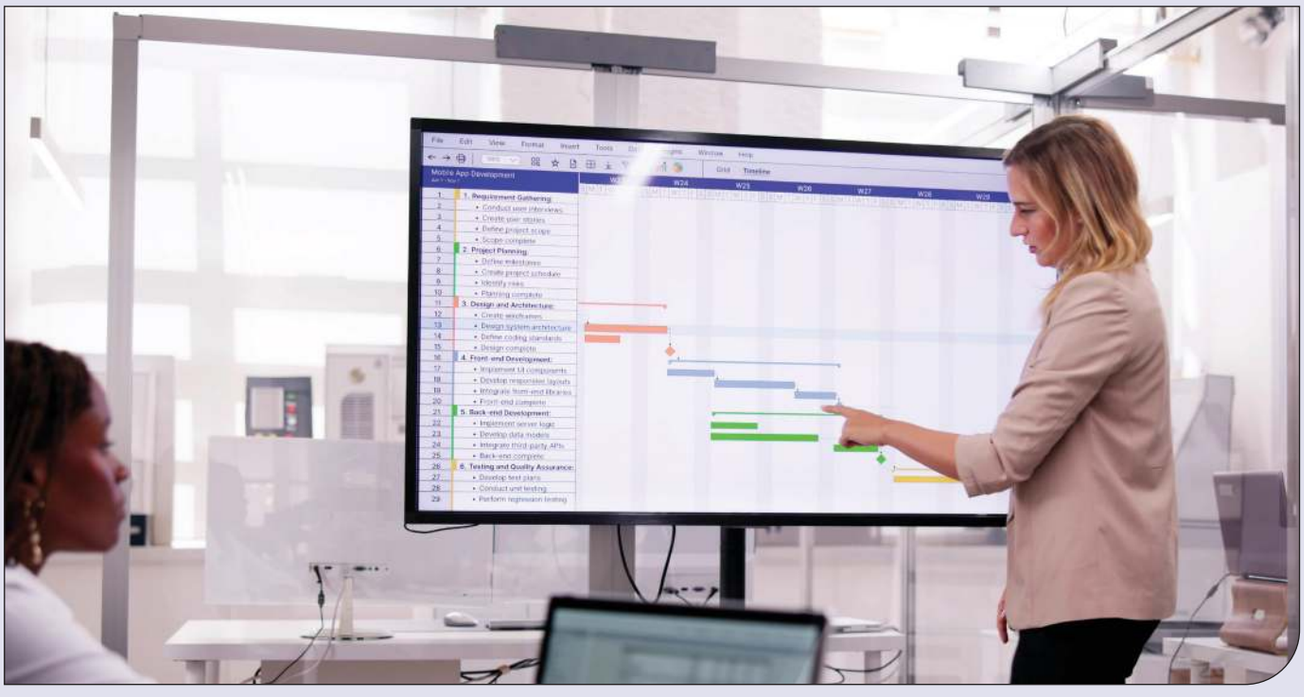


Figure 1.17: Gantt charts give a visual representation of the project schedule, task dependencies, milestones and progress

Table 1.24: Advantages and disadvantages of Gantt charts

Advantages	Disadvantages
• Provide a visual timeline of project tasks, durations and dependencies • Clearly show task dependencies, overlaps and milestones • Help in resource planning, scheduling and tracking progress.	• May oversimplify complex project workflows • Can be challenging to manage for large projects with lots of tasks or changes.

PERT charts

Program Evaluation and Review Technique (PERT) charts are another tool used in project management to visualise and schedule tasks. During the planning phase, PERT charts provide an idea of the time required

to complete a project, which can help with resource allocation and risk management. They are designed to produce estimates of the shortest, longest and most likely time frame for a project. PERT charts are particularly useful for projects with uncertain durations or where several activities can run at the same time.

Features of a PERT chart include:

- **Nodes**: these represent tasks or activities within the project. Each node typically contains the task name, as well as other relevant information such as duration estimates, dependencies and responsible individuals or teams.
- Directed arrows: these connect the nodes in the PERT chart, representing the dependencies between tasks. Arrows indicate the sequence in which tasks must be completed.
- Event points: event points (also called circles or bubbles) represent milestones or key events within the project timeline, such as project start.
- Duration estimates: each task or activity in the PERT chart is associated with how long each task is likely to take, for example: Optimistic (O), Most Likely (M) and Pessimistic (P) durations. These estimates are used to calculate the Expected Time (ET) of each task. Note that, in Figure 1.18, durations are not included as each project will be different.

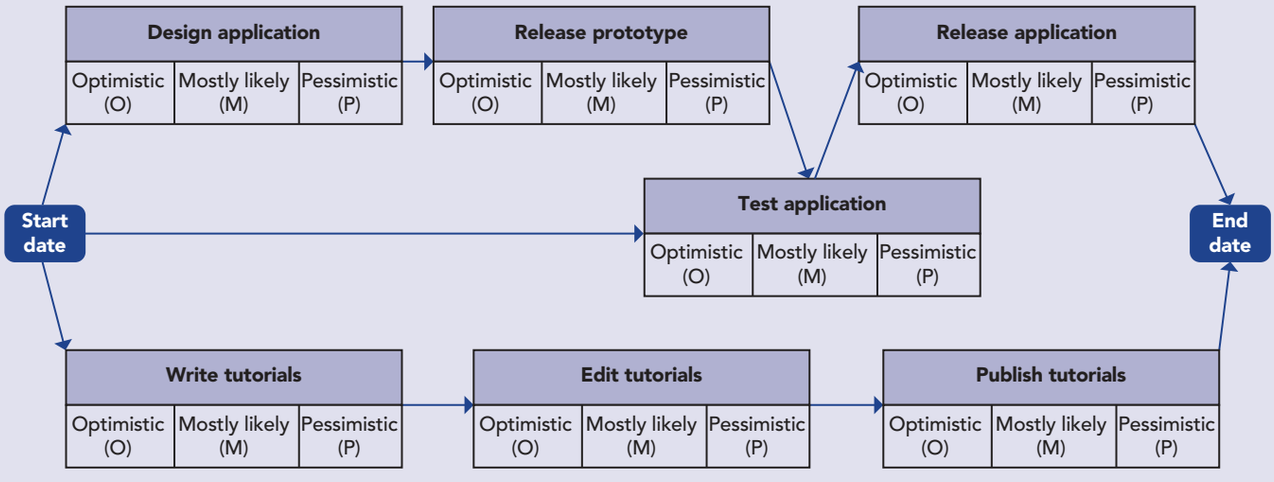


Figure 1.18: PERT is focused on managing uncertainty in project duration

Table 1.25: Advantages and disadvantages of PERT

Advantages	Disadvantages
• Uses estimates for task durations, allowing for uncertainty and variability • Helps in risk analysis by considering optimistic, pessimistic and most likely scenarios • Provides a visual representation of project timelines and dependencies.	• Requires expertise to accurately estimate optimistic, pessimistic and most likely durations for tasks • Can be time-consuming to create and manage, especially for large projects • May not provide as clear a view of the critical path as other tools.

Strengths/Weaknesses/Opportunities/Threats (SWOT) analysis

A **SWOT analysis** (Strengths, Weaknesses, Opportunities, Threats) is a strategic planning tool commonly used in business to assess the internal and external factors that may affect a project or organisation. It can provide valuable insights into various aspects of the project, helping to inform decision-making and planning.

A SWOT analysis is typically shown in a 2×2 grid with a quadrant for each element. Each section is then completed as in Figure 1.19.

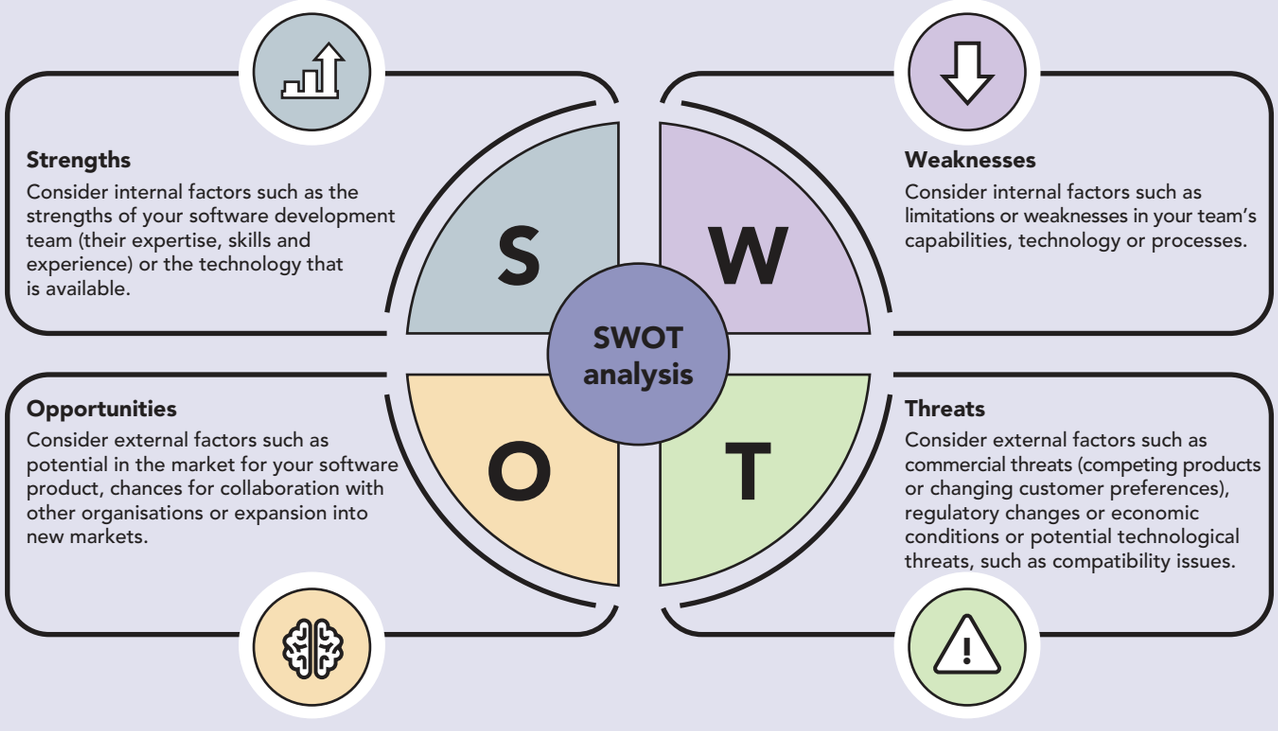


Figure 1.19: A SWOT analysis can help support decision making and develop a business strategy

Once a SWOT analysis has been conducted it can be used to inform project planning in several ways. First, it allows you to develop strategies to address weaknesses and mitigate threats identified in the analysis, for example recruiting additional staff with specific expertise. It can also show you how to make the most of your team's strengths. It can also help set clear project objectives and focus resources and efforts on areas where the project has the greatest potential for success.

Table 1.26: Advantages and disadvantages of SWOT analysis

Advantages	Disadvantages
• Provides a structured approach to identifying internal strengths and weaknesses, as well as external opportunities and threats • Simplifies the analysis of a situation by breaking it down into four key areas • Allows threats to a project to be identified early on • Allows quantitative/ qualitative data to be used to assess a situation.	• Subjective nature may lead to bias • Oversimplification of issues can lead to key aspects being missed • Limited in its ability to prioritise factors identified • Can be difficult to categorise information into the four sections, which can lead to disorganisation.

How defined client requirements affect the selection of project planning tools

Selecting the right project planning tool for the job is important in ensuring success. Defined client requirements play a crucial role in selecting project planning tools as they influence various aspects of project management, including scope, task breakdown, timeline, communication and flexibility.

- **Scope of the project:** the scope of the project, or what needs to be delivered, will influence the selection of an appropriate tool. For example, a large and complex project may be confusing if represented on a flowchart.
- **Task breakdown:** project managers often need to break down a project into manageable tasks that can be easily sequenced. A SWOT analysis or flowchart cannot show tasks and dependencies, whereas a Gantt chart may be more appropriate.
- **Timeline:** client requirements often include deadlines and milestones. Project tools that support the creation of timelines, such as Gantt charts, Critical Path Analysis and PERT charts, allow project managers to allocate resources, set dependencies and establish realistic deadlines based on client requirements.
- **Communication:** client requirements often involve multiple stakeholders with different perspectives and expectations. SWOT analysis and Gantt charts can be useful planning tools to communicate planning and progress to stakeholders whereas some other tools, such as PERT charts, can be more difficult to understand.
- **Flexibility:** client requirements can change over time, so tools that can adapt to reflect these changing requirements can be useful. A Gantt chart or flowchart are easily updated and can flex to changes over time whereas PERT chart are complex and time consuming to make so less able to adapt.

Learning in context 3

Selecting an appropriate planning tool

Amondale Software Solutions, a startup specialising in mobile application development, was getting ready to launch their flagship product: an innovative task management app designed to streamline productivity for users. With a small but dedicated team of developers and designers, they recognised the importance of effective project planning to ensure the successful and timely delivery of their app.

As the project start date approached, the team deliberated on which planning tool would best suit their needs. They considered several options, including Gantt charts, PERT charts and flowcharts. Each tool offered unique features and benefits, but the team needed to choose the one that would provide the most value for their specific project requirements.

They identified several factors that were important in making their decision:

1 Timeline visualisation

2 Resource allocation

3 Stakeholder communication

4 Adaptability.

Timeline visualisation: the Gantt chart provides a clear and visual representation of the project timeline, showing task durations, dependencies and milestones. This allows the team to easily track progress and identify potential bottlenecks or delays. PERT charts can also show a project timeline but require expertise to create them, which the team might not have. A flowchart is limited in its ability to show duration so is not best suited to timeline visualisation.

Resource allocation: with limited resources available, the team needed a planning tool that would help them effectively allocate resources and manage dependencies. The Gantt chart's ability to show task dependencies and resource availability allowed the team to optimise resource allocation and ensure tasks were completed in the most efficient manner. Neither flowcharts or PERT charts show resources so would not be suitable.

Stakeholder communication: the Gantt chart is as a valuable communication tool for the team to share project plans and progress to stakeholders. Its intuitive design and visual layout make it easy for stakeholders to understand the project timeline and milestones. Flowcharts are easy to follow but in a large project are likely to become complicated and harder to navigate. PERT charts can also be complex and difficult to follow, therefore not as suitable for communication as a Gantt chart.

Adaptability: Amondale Software Solutions anticipated that project requirements and priorities might change over time. The flexibility of the Gantt chart allows the team to easily adjust project plans, add or remove tasks, and accommodate changes in scope or timeline as needed. Flowcharts can also be easily updated to reflect changes, although not specifically the timeline of a project. PERT charts can be time consuming to make, so although they can be updated it is not as adaptable as other methods.

Overall, a Gantt chart is the most suitable option for the team. The Gantt chart provides the visibility, control, and flexibility the team needs to navigate the complexities of the project and achieve their goals.

Quick check 7

1 Reflecting on a project that you have been or are going to work on, carry out a SWOT analysis to examine the internal and external factors that might impact the success of your product.

2 Make notes to compare and contrast a Gantt chart and a flowchart as project planning tools. Consider their features and what they may be best used for.

Practice questions 3

When planning an application, relevant legislation should be considered.

1 Explain why considering legislation is important when developing an application. **[2]**

2 A flowchart has been created as part of the development process for a new application.

a Identify **two** advantages of using a flowchart. **[2]**

b Explain **one** disadvantage of using a flowchart. **[2]**

TA4 Application design scoping

Learning intentions

This topic is about how client requirements are determined at the outset of a project and then used to create a specification for development. It will also explore the methods used to decompose or break down the requirements to visualise application designs and when to use them.

It covers:

4.1 Methods of gathering client requirements

4.2 Client requirement specifications

4.3 Decomposition methods

4.1 Methods of gathering client requirements

There are a range of methods that can be used to gather client requirements that vary in the type of information and data that can be collected. These will often be used in combination so that a thorough exploration of the requirements takes place.

Document analysis

Document analysis involves reviewing existing documentation related to the application development project, such as business plans, technical specifications, user manuals or previous project documentation. It helps in understanding the project background, existing requirements, constraints and any other relevant information documented by the client or project stakeholders. The documents help gather data on key requirements, project scope, technical specifications, business goals, constraints and any other relevant information documented in written form. Document analysis is a useful method for gathering clients' requirements if there are similar products on the market or when there is secondary research that shows a gap in client needs.

Table 1.27: Document analysis has its strengths and weaknesses that need to be considered when making choices

Advantages	Disadvantages
• Gives a structured overview of existing requirements, business goals and technical specifications • Can be carried out quickly without needing direct interaction with stakeholders.	• Documents may lack up-to-date information or not cover certain requirements • May not capture the detail around client requirements and preferences.

Focus group

Focus groups involve bringing together a selected group of stakeholders to discuss specific topics related to the application development. This might include members of the target audience for the application, for example. The purpose is to gather insights, opinions and feedback from multiple perspectives simultaneously, fostering discussion and idea generation. Focus groups gather information on the perspectives, preferences

and opinions on specific features, functionalities or aspects of the software application. Focus groups are useful when exploring new ideas and there is a need for early feedback on pain points with existing solutions for end users.

Table 1.28: Focus groups have their strengths and weaknesses that need to be considered when making choices

Advantages	Disadvantages
• Gives insights from multiple stakeholders at the same time so a range of perspectives can be given • Facilitates consensus-building (agreement) through group interaction.	• Some participants may dominate the focus group and control the discussion so not everyone is heard • People in the group might conform to the majority and not be willing to give opinions that disagree with this view.

Interviews

Interviews involve conducting one-on-one discussions with individual stakeholders, such as clients, end-users or subject matter experts for the application development project. The purpose is to gather detailed information, preferences, needs and expectations directly from stakeholders in a more personalised and in-depth manner. Through carefully framed questions, the interviewer will gather detailed information on the requirements, specific use cases, potential issues, individual perspectives and any other relevant insights provided by the interviewees. Interviews are useful in the early stages of gathering requirements as they provide an opportunity to clarify or ambiguous or conflicting requirements.

Table 1.29: Interviews have their strengths and weaknesses that need to be considered when making choices

Advantages	Disadvantages
• Provides detailed and personalised understanding of individual stakeholder needs, preferences and concerns • Allows for probing and follow-up questions to clarify responses and explore topics further.	• Requires time and effort to conduct interviews with multiple stakeholders • Interviewer bias or leading questions may influence responses and distort the collected data.

Meetings

Meetings are scheduled sessions involving project stakeholders to discuss various aspects of the application development. These may take place in person or online. The purpose of a meeting is to ask a range of open questions to fully understand the project requirements and support communication and collaboration. Meetings are useful for checking the understanding of requirements and aligning expectations between stakeholders.

Table 1.30: Meetings have their strengths and weaknesses which need to be considered when making choices

Advantages	Disadvantages
• Enables real-time communication, collaboration and alignment of expectations among stakeholders • Allows stakeholders to ask questions and seek clarification if something is not clearly understood.	• Meetings can be time-consuming, especially if not well-structured or focused • Meetings usually have a limited number of attendees so it would not be representative of a wide range of opinions.

Observation

Observations involve directly observing users interacting with existing systems or workflows relevant to the application. The purpose is to understand user behaviours, tasks, workflows, issues and opportunities for improvement by witnessing real-world usage. By observing the use of the existing application or systems, information can be collected on user behaviours, inefficiencies, usage patterns and any other insights gained from observing users in their natural environment. This means that they are useful when developing early ideas on User Experience (UX) and User Interfaces (UI).

Table 1.31: Observation has its strengths and weaknesses that need to be considered when making choices

Advantages	Disadvantages
• Provides first-hand insights into how users interact with existing systems or workflows in real life situations • Reduces chance of bias as observations are based on actual behaviour rather than someone reporting their own activity.	• May not collect information on the thoughts or emotions of users so limited in deeper understanding • Users may change their behaviour when aware of being observed, potentially affecting the observation.

Problem reports

Problem reports are records of issues, bugs or complaints faced by users or stakeholders in existing software applications. The purpose of examining problem reports is to identify and document issues affecting the user experience or functionality of the software, which need to be addressed in the development process. This analysis might help scope client requirements for the development of a new application. Information gathered can include the details of problems encountered, how often they occur, the impact on user experience, potential solutions and any other relevant information related to reported issues. Problem reports are useful in the early stages of development as they highlight existing issues or pain points experienced by users or stakeholders with the current system or process.

Table 1.32: Problem reports have their strengths and weaknesses which need to be considered when making choices

Advantages	Disadvantages
• Helps identify existing issues and bugs experienced by users, guiding improvement efforts • Provides specific feedback on areas needing attention, facilitating targeted solutions for future development.	• Users may only report issues that are particularly frustrating for them, leading to a skewed representation of problems • Problem reports may lack context or detailed information necessary for effective resolution in future development.

Questionnaire

Questionnaires are a series of open and/or closed questions that are used to collect feedback from a large group of stakeholders. These may be conducted online or on paper. They gather both quantitative (numerical) and qualitative (descriptive) data that can be analysed to help in the develop of client requirements. This might include numerical data around application usage, descriptive feedback on specific aspects of the application or demographic information. Questionnaires are useful when for gathering data from a large group or those who