Software Engineering
SEN152

Note

It is important to note that this study guide must be read in conjunction with the study material contained on the module course site accessed via your Learning Management System (LMS), Canvas.

Please consult Canvas to confirm whether a prescribed textbook must be purchased. Where necessary we will refer to specific pages or chapters.

There may also be reference to additional recommended reading material available for free or at a cost. This will be optional reading intended to enhance your understanding of the material.

The content of the STADIO study guides and teaching documents are not intended to be sold or used for commercial purposes. Such content is, in essence, part of tuition and constitutes an integral part of the learning experience, regardless of the mode.

Links to websites and videos were active and functioning at the time of publication. We apologise in advance if there are instances where the owners of the sites or videos have terminated them. Please contact us in such cases.

A Glossary of terms may be provided at the end of this study guide.

Any reference to gender includes all genders. Similarly, singular may refer to plural and vice versa.

Where AI tools were used to present and organise content for optimal accessibility to students, rigorous quality assurance processes were adhered to.

All figures, tables and other visuals included in this study guide were specifically created for STADIO by commissioned authors, unless otherwise indicated.

It is your responsibility to regularly access Canvas to make sure that you always refer to the latest and most updated material for this module.

We encourage students to make use of the available resources on the STADIO Online Library available on Canvas.

Table of Contents

Module Purpose and Outcomes

Module purpose

The Software Engineering module provides a solid foundation in the discipline of developing and maintaining software that are efficient and reliable, are affordable to develop and maintain, and satisfy the customer requirements. This module provides a history of software development in in the organisational context.

Students will acquire broad knowledge about the Software Engineering process and the range of methods, tools, and techniques utilised, illustrated by means of a variety of case studies. Students will acquire skills in the practical application of methods and techniques and the use of software development platforms and tools. Students will also learn the importance of quality management in the software development process.

Module outcomes

Upon successful completion of this module, the student will be able to:

1. Demonstrate insight in the history of software development.
2. Demonstrate an understanding of different options for software development life cycles related to software design and development, and key terms, facts, principles and rules of software development.
3. Apply methods, tools and modelling techniques commonly employed during the various phases of software development.
4. Demonstrate an understanding of the importance of software maintenance and documentation.
5. Identify modern software development and management platforms, tools, and services, and outline the nature of the support provided.
6. Demonstrate an understanding of the notion of quality in software and how a Quality Management System can provide the required organisational framework.

Prescribed reading

Letaw, L. 2024. Handbook of Software Engineering Methods. 2nd Edition. Online available: https://open.oregonstate.education/setextbook/ [Accessed: 07 February 2025].

Recommended reading

Agile Alliance. 2025. The Agile Manifesto. Online available: https://www.agilealliance.org/agile101/the-agile-manifesto/ [Accessed: 12 March 2025].

ISO. 2025. Online available: https://www.iso.org/search.html?PROD_isoorg_en%5Bquery%5D=software%20quality [Accessed: 07 April 2025].

OMG. 2025. About the Unified Modeling Language Specification Version 2.5.1 Online available: https://www.omg.org/spec/UML [Accessed: 26 March 2025].

Sommerville, I. 2011. Software Engineering 9th edition. Online available: https://uoitc.edu.iq/images/documents/informatics-institute/exam_materials/Software%20Engineering%20(9th%20Edition)%20by%20Ian%20Sommerville.pdf [Accessed: 07 February 2025].

TOPIC 1
The History of Software Engineering

1. INTRODUCTION

This topic relates to the following module outcome:

1. Demonstrate insight in the history of software development.
2. Demonstrate an understanding of different options for software development life cycles related to software design and development, and key terms, facts, principles and rules of software development.

In this topic, you will gain knowledge in the following areas:

1.1 The software engineering journey.

1.2 The software engineering process.

1.3 Software development principles.

Prescribed reading

Letaw, 2024:

- What's Software Engineering? (Page 1)
- What's the Purpose of Software Engineering? (Page 2)

1.1 THE SOFTWARE ENGINEERING JOURNEY

1.1.1 1940s – 1960s

The era of early computers, such as ENIAC (Electronic Numerical Integrator and Computer) and UNIVAC (Universal Automatic Computer), is where the roots of software engineering can be found. During the 1940s to 1950s, hardware development included programming tasks. Machine-level instructions were used to build computer programs. From the 1950s to the 1960s, code was written more efficiently by using languages such as COBOL (Common Business-Oriented Language) (Khan, 2023).

Figure 1 shows an old photograph of the world's first computer, the electronic numerical integrator and calculator (ENIAC). According to Swaine & Freiberger (2025) it was built in 1943 (completed in 1946) by the United States of America during World War II by the School of Electrical Engineering at the University of Pennsylvania. It was said to be programmable even if it took days to rewire it for every problem. Swaine & Freiberger (2025) states that "it could execute different instructions or alter the order of execution of instructions based on the value of some data. (For instance, IF X>5 THEN GO TO LINE 23)". The computer had flexibility and the potential to be used for other problems in the future.

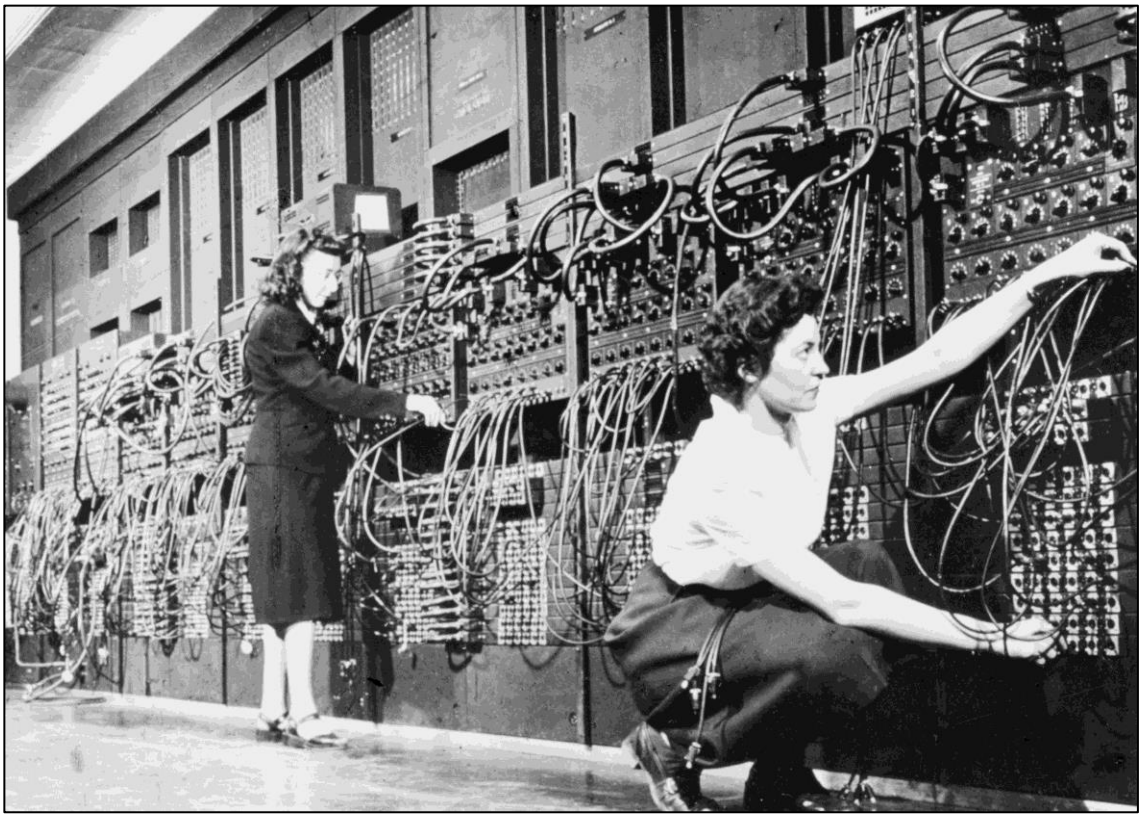

Figure 1 – Photograph of World's First Computer, the Electronic Numerical Integrator and Calculator (National Archives, 2025)

1.1.2 1960s – 1990s

Software Engineering emerged as a study discipline. Margaret Hamilton, an MIT computer scientist, invented the term 'software engineering'. Margaret was involved in the Apollo space program's software development projects. The need for sound methodologies and structure emerged as software engineering projects became more complex over time. During the 1970s, software engineering methodologies emerged. The Waterfall model became popular. Programming in

modules also became popular, as scientists realised that modules could be built and reused to save time (Khan, 2023).

Iterative and incremental development started to take shape in the 1980s and models like the Spiral model and Rapid Application Development (RAD) were used. Object-oriented programming (OOP) became more popular during this time (Khan, 2023).

1.1.3 1990s – present

The internet and personal computing took form and software applications expanded. Agile methods gained a lot of attention during this time. Open-source development, supporting the sharing of software code, was a growing field in software development (Khan, 2023).

Software engineering advancements in the 21st century included the unification of Information Technology (IT) operations and software development. The business focus was on collaboration and process automation. The influence of cloud computing, machine learning and Artificial Intelligence (AI) on software applications created more sophisticated solutions (Khan, 2023).

These solutions support organisations working towards gaining a competitive advantage in any industry. IT innovation is an ongoing process, and organisations will continue to evolve and develop with the support of high-end software solutions.

1.1.4 Computer programming languages – a timeline

The following timeline showing the rise and adoption of programming languages was compiled by HP Tech Takes (2018):

- **1949**: Assembly language was applied as a programming language. Machine code language was simplified to task a computer with what to do.
- **1952**: Autocode was developed by Alick Glennie. This was the first compiled computer programming language used and was converted directly into machine code.
- **1957**: FORTRAN was developed by John Backus. This language was applied specifically for mathematical, scientific, and statistical projects.
- **1958**: An algorithmic language, Algol, was developed. Algol was a precursor to languages like C and Java.
- **1959:** Dr. Grace Murray Hopper created COBOL, which could operate on all computer types.

- **1959**: LISP was created by John McCarthy, which is still applied today. It was used in artificial intelligence (AI) research tasks and presently can be applied with Ruby and Python.
- **1964**: BASIC was developed by John G. Kemeny and Thomas E. Kurtz for students lacking strong math technology skills. This enabled the students to use the computers.
- **1970**: Pascal was developed by Niklaus Wirth. Apple used this language in its early software development projects.
- **1972**: Smalltalk was created by Alan Kay, Adele Goldberg, and Dan Ingalls. Smalltalk assisted programmers in updating code faster.
- **1972**: C was created by Dennis Ritchie. C is seen as the first high-level programming language (less like machine code and more like human language).
- **1972**: SQL was developed for IBM by Donald D. Chamberlin and Raymond F. Boyce. This database language enabled developers to update data in databases.
- **1978**: MATLAB was created by Cleve Moler to create mathematics programs in the fields of education and research.
- **1983**: Objective-C was developed by Brad Cox and Tom Love which was applied by Apple for software development.
- **1983**: C++ (an extension of the C language) was developed by Bjarne Stroustrup. This language is most often applied globally.
- **1987**: Perl, a scripting language, was created by Larry Wall
- **1990:** Haskell was created to process complex math calculations.
- **1991**: Python, a simplified language, was developed by Guido Van Rossum
- **1991**: Visual Basic was developed by Microsoft developed. This language allowed programmers to use a drag and drop technique to update code.
- **1993**: R was created by Ross Ihaka and Robert Gentleman for data analysis tasks conducted by statisticians.
- **1995**: Sun Microsystems created Java, which could be applied in hand-held devices.
- **1995**: PHP for web development, was created by Rasmus Lerdorf. It is still in use today.
- **1995**: Ruby, an all-purpose language used in web applications, was created by Yukihiro Matsumoto
- **1995**: Brendan Eich created JavaScript to improve interactions in web browsers.
- **2000**: Microsoft created C# (similarities to Java) as a mixture of Visual Basic and C++.
- **2003**: Martin Odersky developed the programming language, Scala.
- **2003**: Groovy (as a result of Java) was created by Bob McWhirter and James Strachan

- **2009**: Go was developed by Google to solve problems with large systems.
- **2014**: Apple created Swift to replace Objective-C, C, and C++.
- **Today**: Many older programming languages are still used today, and many are used as a foundation for the creation of new or more advanced languages. The aim is to reduce the workload on programmers and to simplify software creation.

An article published on Linkedin by Sharma (2024), lists the five best programming languages for application development in 2025:

1. JavaScript
2. Java
3. Swift
4. Kotlin
5. Dart

Activity

Read https://developer.apple.com/swift/ and conduct brief research on Apple's Swift language.
Time allocation: 10 minutes

1.1.5 Integrated Development Environments (IDEs)

Integrated Development Environments (IDEs) are tools used by software developers to streamline their design and programming experience. An IDE has an integrated user interface which allows developers to easily create, update and debug code. It automates many coding tasks which speeds up the development process (Okeke, 2022).

There are many benefits of IDEs. Some of these benefits are briefly described by Okeke (2022):

- IDEs support the development cycle and streamlines the process.
- IDEs check for errors automatically and this improves code quality.
- Developers can complete code much faster with the capabilities of IDEs.
- An IDE provides a centralised space for developers. Version control and debugging tools help developers manage their work more efficiently.
- Developers can quickly and efficiently update and rename code.

Activity

Read https://www.techrepublic.com/article/best-ide-software/ and study the classifications of IDEs.
Time allocation: 10 minutes

1.2 THE SOFTWARE ENGINEERING PROCESS

1.2.1 Software engineering in the organisation

People care about business problems, and these problems can be solved with software engineering to create feasible and sustainable software solutions. Software Engineering is defined as the "systematic application of scientific and technological knowledge, methods, and experience to the design, implementation, testing, and documentation of software" (Letaw, 2024).

An initial concept is needed for a software project to start. There might be an unmet need in an organisation or a specific department. A list of required functionalities could trigger the need for a new systems development project. Organisations will determine whether a solution is available off-the-shelf or whether the solution needs to be built in-house. This will depend on the resources and skills of the organisation (Allbee, 2018). Organisations that do not possess the required IT functions to support systems development, will outsource the systems development project to a third-party IT company.

There is no one set of global methodologies or techniques which could be adopted for software development as software systems come in various sizes and types (Sommerville, 2009). According to Lee (2013), software engineering is the "process of designing, developing and delivering software systems that meet a client's requirement in an efficient and cost-effective manner".

Software Engineering is not just about writing lines of code while applying development methodologies. Software engineers should have a range of skills, including soft skills (Allbee, 2018).

Sommerville (2011) states that "a professionally developed software system is often more than a single program. The system usually consists of a number of separate programs and configuration files that are used to set up these programs. It may include system documentation, which describes the structure

of the system; user documentation, which explains how to use the system, and websites for users to download recent product information".

Modern software becomes more complex as time goes on. This highlights a need for proper development plans, sound methodologies and clear communication plans to be in place. It is important for organisations to understand the level of complexity of any software solution to deliver it successfully (Lee, 2013).

1.2.2 Software engineering participants

Computer programmers were thought to be the only main participants in software engineering, but a broad range of skills are required throughout a project's lifetime. Participants from various fields are required to oversee a software project (Lee, 2013).

A software development project today could include systems analysts, project managers, business analysts, system architects, programmers, testers, senior management, end-users, or even outsourced human resources if required, for example a risk management consultant for a high-risk project. The customers and sponsors are involved, as are any role players whose departments are affected by the system. The term 'stakeholder' requires the project manager to include all who are directly and indirectly affected by the new software development project.

Managers working on different levels in the organisation require different skill sets. Three categories of managerial skills are interpersonal, technical, and administrative and conceptual skills. Higher management might be more concerned with administrative and conceptual skills, as they are responsible to understand the bigger picture and how the software project fits into the business strategy and vision (Letaw, 2024).

1.2.3 Cloud computing and Artificial Intelligence (AI)

Organisations in the past had to create and maintain their own software services. Cloud computing providers have changed the way organisations manage its software operations. Many organisations save costs by moving the responsibility of software operations to cloud providers for a fee. The risks associated with the maintenance of software services lie with the service provider and the organisations can focus their time and energy on other work in their respective industries.

In the past, small businesses didn't own their own mainframes as it was too expensive. These businesses would then utilise computing capabilities of bigger businesses that did own mainframes. The concept of buying software services as a utility is not a new phenomenon (Bigelow, 2025).

Cloud computing is critical in households and businesses. Working remotely is one example of how cloud computing has changed the face of employment. Examples of cloud services are Microsoft Azure, Google Drive, Dropbox and Apple iCloud. This reduces the need to use flash sticks and offers a much more secure way to manage and store data (Bigelow, 2025).

Cloud computing service types include the following (Microsoft, 2025):

Infrastructure as a Service:

Compute, network and storage resources are provided on demand (pay-as-you-go). Organisations do not have to maintain physical data centres and changes to infrastructure are easy to apply (Microsoft, 2025).

Platform as a Service (PaaS):

This includes a complete software development and deployment environment residing in the cloud. It includes storage, servers an networking capabilities, and also development tools, database systems and much more (Microsoft, 2025).

Software as a Service (SaaS):

Users can connect to and use Internet cloud-based applications. Many organisations (like STADIO) use Microsoft Office 365 for email, office tools, and meeting calendars. These organisations pay the service provider on a pay-as-you-go basis and do not have to manage the hardware and software as this is the responsibility of the service provider (Microsoft, 2025).

Serverless:

According to Microsoft (2025), "with serverless applications, the cloud service provider automatically provisions, scales, and manages the infrastructure required to run the code". The name 'serverless' originates from infrastructure provisioning and management tasks that are kept invisible to developers even though the servers are still running all the code.

The future of cloud computing will include businesses taking a strategic approach to invest in cloud services. An increase in operational efficiency and advanced security protocols in cloud infrastructure are also emerging. Quantum computing is another term deserving attention as it seeks to increase the speed and complexities of data operation. Lastly, cloud providers are expected to protect

the future of the environment by looking at how they use energy, their carbon footprint and possibilities for investing in renewable energy (Bigelow, 2025).

Artificial intelligence (AI) is defined as "technology that enables computers and machines to simulate human learning, comprehension, problem solving, decision making, creativity and autonomy" (Stryker & Kavlakoglu, 2024).

Generative AI is a popular term in today's business world. These technologies can create original content, including videos, images and text. Generative AI tools are built by adopting machine learning (ML) and deep learning. Figure 2 shows the history and relation between these concepts (Stryker & Kavlakoglu, 2024):

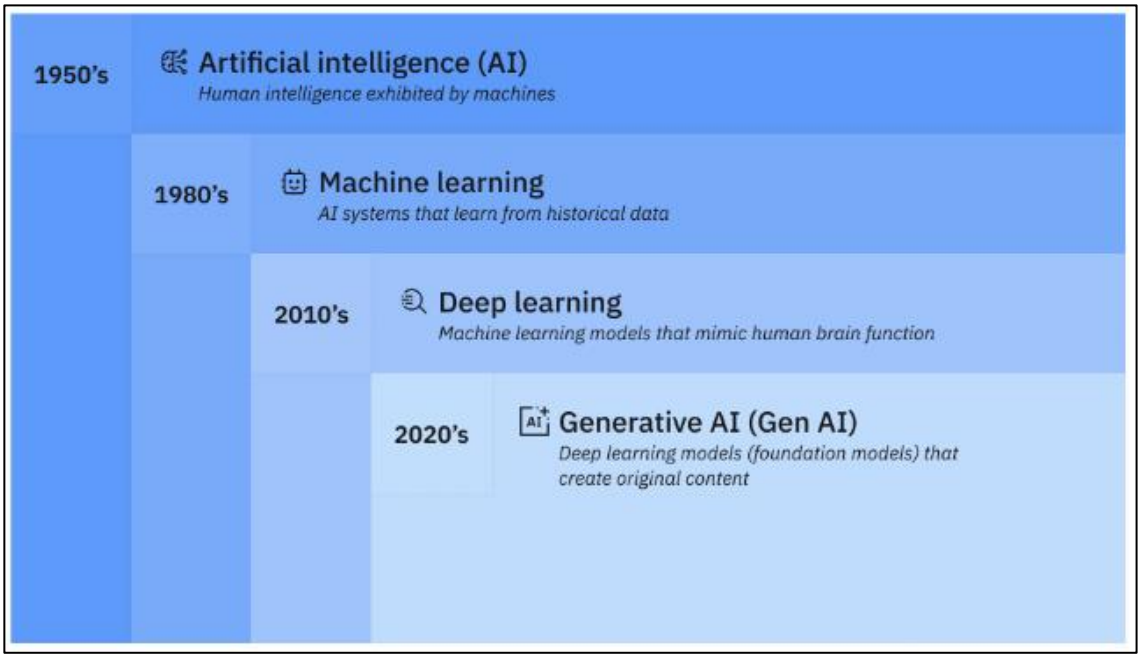

Figure 2 – Relation between AI, ML, deep learning and Gen AI (Stryker & Kavlakoglu, 2024)

The development of software solutions employing Artificial Intelligence is on the rise. Many organisations are competing to achieve breakthroughs in the AI field. The investment potential of AI powered solutions are great as it influences many industries.

1.3 SOFTWARE DEVELOPMENT PRINCIPLES

1.3.1 Principles for developers

The world depends on technology and organisations need to invest in software solutions to remain competitive in their respective markets. Software development in general is fuelling innovation in industries.

High quality software is software that meets the user's needs and includes all the required functionalities. Good software should be easy to maintain and usable. (Sommerville, 2011).

A principle is a rule invented to follow to achieve a positive outcome. Jewel (2023) wrote a blog post discussing software development principles for developers that should be practised daily:

DRY (Don't Repeat Yourself)

The aim is to reduce duplicated code and avoid repetition of work. Developers could save time by creating modules for reuse in future projects (Jewel, 2023). An example would be to reuse a login code module across various solutions.

KISS (Keep It Simple, Stupid)

The aim is to keep software simple and easy to maintain. This ensures that solutions are scalable and not overly complex.

YAGNI (You Ain't Gonna Need It)

Developers could easily code future needs into solutions but should be aware that this could introduce bugs or add unwanted complexities.

SOLID Principles

SOLID lists design principles for scalable software that is easy to maintain. Each letter represents a design principle.

- Single Responsibility Principle (SRP)
- Open/Closed Principle (OCP)
- Liskov Substitution Principle (LSP)
- Interface Segregation Principle (ISP)
- Dependency Inversion Principle (DIP)

Continuous Integration and Continuous Deployment (CI/CD)

The development process should be automated. Code integration, code testing and deployment will benefit from automation. Software development cycles are shortened, bugs are fixed faster, and the speed of development work increases.

Gupta (2024) states that to improve the quality of software, the following principles and rules should be followed to provide theoretical and practical strategies for development work:

- **Architecture first approach:** In the early phases of software development, create a robust architecture to improve project productivity.
- **Iterative life cycle process:** Stages in development are repeated and refined to improve software quality.
- **Component-based approach:** Modules are reused to speed up the development process, to improve consistency and software maintenance and upgrades.
- **Change management system:** This improves how changes are managed while improving software stability and maintenance.
- **Round trip engineering:** Reverse engineering and code generation is integrated to enhance overall efficiently.
- **Model-based evolution:** Models create a conceptual framework to understand the behaviour of software. The software design and functionality are improved as developers receive real-time feedback while meeting user requirements.
- **Objective quality control:** Quality is important in software development. Quality checklists are monitored, and quality is continuously improved.
- **Evolving levels of details:** Use cases and design details are incrementally refined to improve software flexibility.
- **Establish a configurable process:** Customisation is improved by adapting current methodologies, techniques and tools to different projects.
- **Demonstration-based approach:** Developers showcase software to the stakeholders. Stakeholder feedback could be valuable throughout the development process, and this will enhance product quality.

Activity

Read the section titled "Real-life examples of companies using principles of software development" in the article by Gupta (2024): https://www.turing.com/blog/principles-of-software-development-guide
Study any one example principle discussed in the article.
Time allocation: 10 minutes

Summary

The study of software engineering is broad and applies to many industries. The history of software engineering confirms the long journey of this field and how rapidly it has evolved over the years. Many organisations complete software development projects aligned with their overall business strategy to leverage opportunities for competitive advantage.

Software development principles are valuable for developers to improve the quality of their work. It is important for organisations to ensure that software development principles are aligned with the organisational and business strategies

Cloud computing and artificial intelligence (AI) have a profound impact on the way software is developed. The emergence of generative AI and the speed at which organisations are competing in this field, is a testament to the evolving nature of software engineering.

Self-Assessment Questions

1. Describe the emergence of the field of software engineering.

2. Discuss why older programming languages are still relevant today.

3. Explain why software developers need sound methodologies.

4. Describe the differences between machine learning and deep learning.

5. Discuss the importance of 'reuse' as a software development principle.

6. Discuss the impact of cloud computing on software development.

TOPIC 2
The SDLC, methodologies and software development platforms and tools

2. INTRODUCTION

This topic relates to the following module outcome:

2. Demonstrate an understanding of different options for software development life cycles related to software design and development, and key terms, facts, principles and rules of software development.
3. Apply methods, tools and modelling techniques commonly employed during the various phases of software development.
4. Demonstrate an understanding of the importance of software maintenance and documentation.
5. Identify modern software development and management platforms, tools, and services, and outline the nature of the support provided.

In this topic, you will gain knowledge in the following areas:

2.1 The System Development Life Cycle (SDLC)

2.2 The Waterfall model

2.3 Agile methodologies

2.4 Software development platforms and tools

Prescribed reading

Letaw, 2024:

- The Software Development Life Cycle (Page 7)
- Agile, Scrum, and Agile Methods (Page 10)

2.1 THE SYSTEM DEVELOPMENT LIFE CYCLE (SDLC)

Systems analysts utilise the Systems Development Life Cycle (SDLC), a structured process, to create information systems. While each organisation may adapt the SDLC to fit its specific needs, the fundamental approach remains consistent. This process progresses through a series of stages, ultimately resulting in a high-quality, efficient information system. This process is shown in Figure 3. Over time, an information system will inevitably become obsolete, necessitating an upgrade or improvement. At this point, the SDLC methodology is applied to enhance the existing system or implement a new one to address business challenges or opportunities. If standard methodologies do not fully align with their requirements, some organizations develop customized approaches based on the SDLC framework.

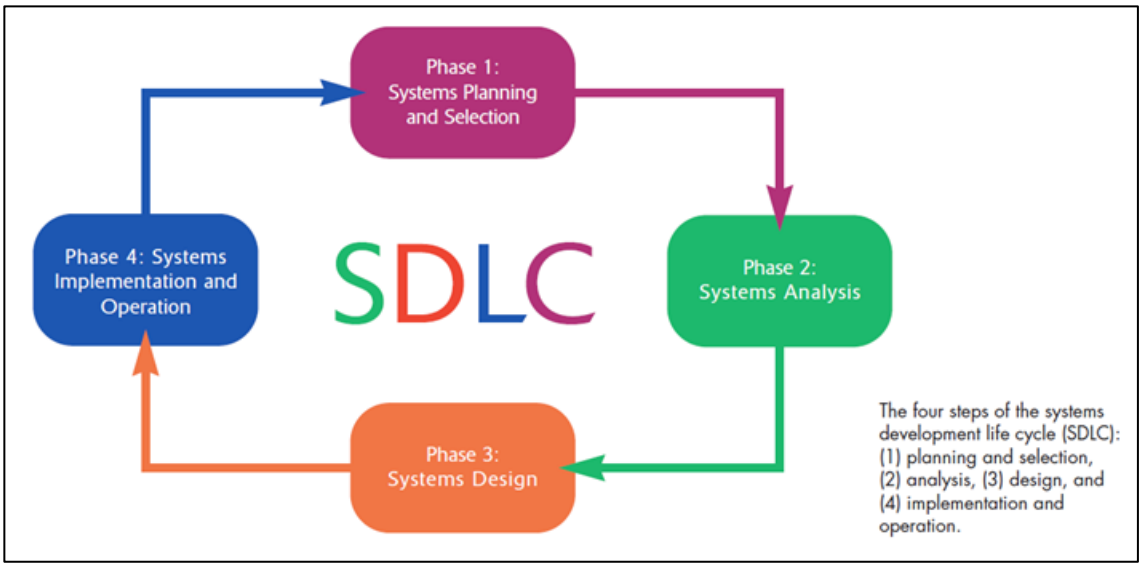

Figure 3 – The SDLC (Valacich, George & Hoffer, 2015:39)

Stage 1: System planning and selection

The information systems department receives numerous system requests as individuals and departments recognize the need for new or improved systems. A systems analyst is responsible for evaluating both existing and required information systems within an organization (Valacich *et al.,* 2015). The analyst must prioritize potential development projects and identify the most feasible one to pursue. Defining the project scope is crucial, and a feasibility analysis is typically conducted, often incorporating a SWOT analysis to assess the system's strengths (S), weaknesses (W), opportunities (O), and threats (T). Based on this evaluation, a recommendation is made on whether the proposed project should proceed.

Stage 2: System Analysis

In this phase, the systems analyst examines the existing system to understand the issues the new system should address. The goal is to develop logical models that define "what" the system should accomplish, rather than "how" it will function (Valacich *et al.,* 2015). Gathering requirements from stakeholders is essential, followed by structuring, analysing, and selecting these requirements to establish the system boundary—clarifying what is included and excluded from the system (IBM, 2024).

Stage 3: System Design

Systems design translates requirements into a detailed blueprint for development. This phase involves creating both logical and physical design specifications (Valacich *et al.,* 2015). The logical design defines "what" the system should do, while the physical design focuses on "how" it will be implemented. During the physical design stage, a technical specification document is created, outlining input and output requirements, user interfaces, system architecture, and other key elements needed for the next phase. According to IBM (2024) prototypes could be created to obtain stakeholder feedback.

Stage 4: System Implementation and Operation

In this phase, the system is developed and coded based on the design specifications. Rigorous testing is conducted to identify and resolve errors, ensuring the system meets quality standards. End-users receive training, and system documentation—such as user manuals and technical guides—is provided. Additionally, a conversion plan is executed to ensure a smooth transition from the old system to the new one (Valacich *et al.,* 2015).

Systems analysts use the SDLC to create information systems. This process involves exploring both traditional methodologies, such as the Waterfall model, and agile models and techniques that can be applied during systems development projects.

Sequential development refers to a method where each phase flows into the next, and a phase cannot begin until the previous one is completed. This approach is often referred to as the traditional or Waterfall development approach.

2.2 THE WATERFALL MODEL

There are many variations of the Waterfall model still in use today. The Waterfall model typically consists of six phases (refer to Figure 4):

1. Project initiation
2. Analysis
3. Design
4. Development
5. Implementation
6. Review and closure

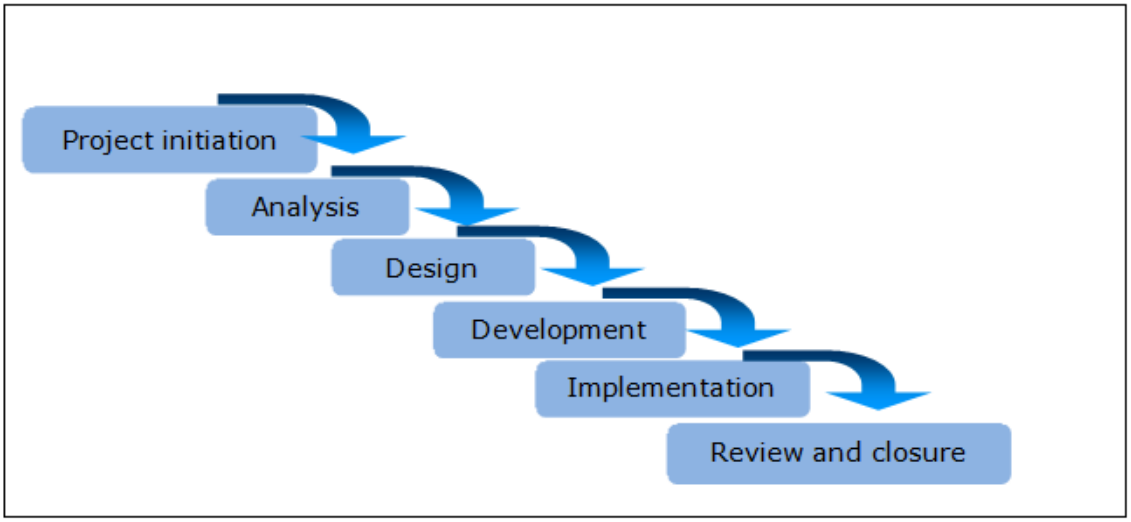

Figure 4 – The Waterfall model (Visagie, 2025)

Figure 5 shows another version of the Waterfall model by Awati & Gillis (2024):

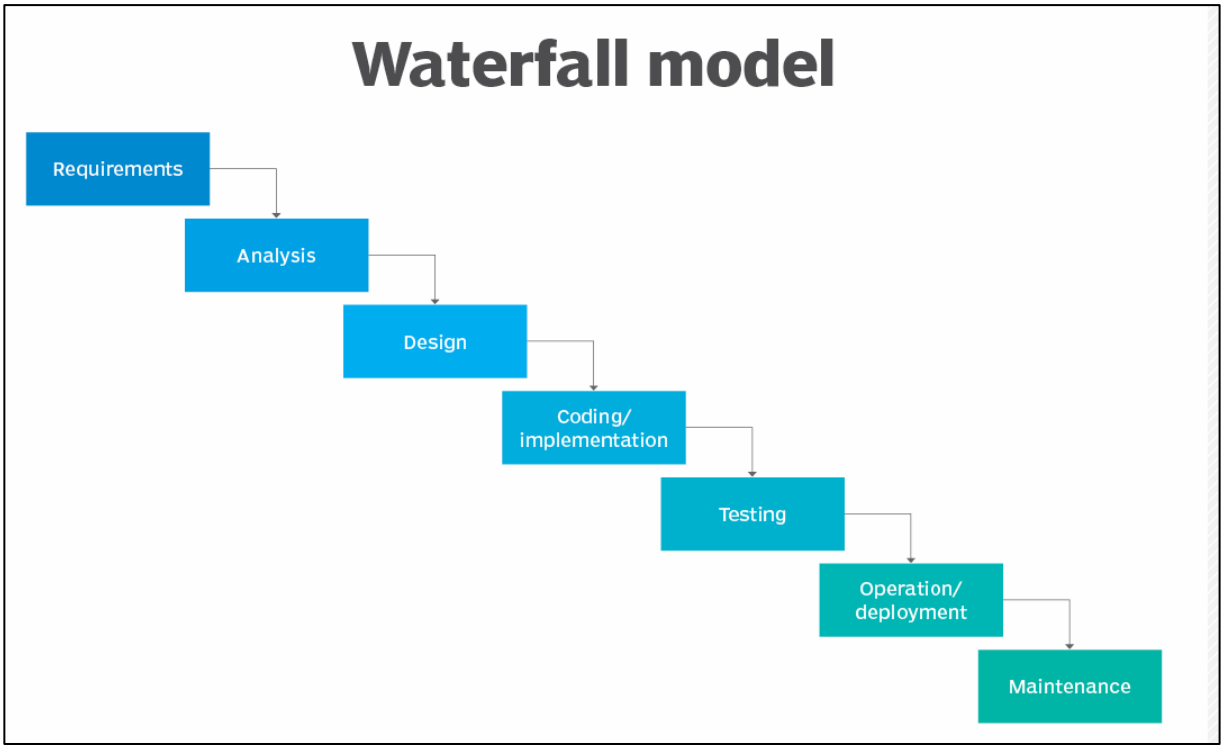

Figure 5 – The Waterfall model (Awati & Gillis, 2024)

Each completed phase in the Waterfall model results in a tangible product or deliverable, which serves as a milestone to assess the likelihood of a successful outcome. The next phase cannot begin until the previous one has been successfully completed (Atlassian, 2025).

This model is ideal when the requirements are clearly defined and stable, when structure and control are necessary during the development process, and when comprehensive system documentation is crucial.

If changes are introduced later in the development process, it can be time-consuming and costly to revisit previous phases, as this often requires significant rework. The Waterfall model is best suited for situations where (Atlassian, 2025):

- Requirements are well-defined and unlikely to change
- The product definition is stable
- The technology is well-understood
- A new version of an existing product is being developed
- Porting an existing product to a new platform

While the Waterfall model is simple and effective, it has shown to be highly inflexible. Flexibility, however, is a key feature for managers, who prefer using deliverables and milestones to track the progress of development according to a

set schedule. Additionally, many systems follow a sequential process, making the Waterfall model a natural fit for such projects.

2.3 AGILE METHODOLOGIES

2.3.1 Agile Alliance

Agile software development emerged in response to the limitations of traditional methodologies. In the late 1990s, there was a growing need for more flexible methodologies that could better handle changing requirements. In 2001, the Agile Manifesto was created by a group of programming methodology experts who formed the Agile Alliance. The manifesto outlines four core values, stating that "while there is value in the items on the right, we value the items on the left more" (Agile Alliance, 2025):

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

2.3.2 Prototyping

Prototyping is an iterative process which involves creating a scaled-down version of a system that is analysed, designed, developed, and tested. This approach allows for early user feedback during the development process, enabling developers to make improvements as soon as they are identified, ensuring a high-quality end-product. Prototyping is an iterative process, where the system is developed in small cycles, and user feedback is incorporated into each new iteration. This fosters close collaboration between developers and users throughout the development process.

The prototyping approach requires a close working relationship between users and designers. The following are benefits of prototyping (Whitten & Bentley, 2007:598):

- It encourages and requires active end user participation.
- It involves iteration and accommodates change.
- It endorses the philosophy that end users will not know what they want until they see it.
- Prototypes are active models that end users can experience with their senses.
- Errors can be detected much earlier.

- It can lead to better solutions and increase creativity, as prototyping allows for rapid user feedback.

The following are drawbacks of prototyping (Whitten & Bentley, 2007:598):

- It encourages a focus on a cycle of coding, implementation and repair.
- It does not eliminate the systems analysis phase.
- It does not address all design issues, some of which can be forgotten if care is not taken.
- It often leads to premature commitment to a specific design.
- During prototyping, systems scope and complexity can rapidly increase beyond the original plan.
- It can reduce design creativity.

Investing in prototyping can significantly enhance efficiency and reduce costs for companies by allowing them to test concepts with end-users and gather valuable feedback. This approach enables developers to evaluate ideas before full-scale production, increasing the likelihood of user acceptance for the final product.

Among the various mobile application prototyping tools available today, Axure stands out. It is extensively utilized by corporate entities for prototyping during software development projects. Axure's capabilities include interactive prototyping that enhances visualisation, facilitates efficient iteration, and supports user testing and feedback collection. This tool allows designers to simulate realistic interactions, enabling users to navigate prototypes and provide insights that help refine usability early in the design process (Axure, 2025).

Axure promotes collaboration among teams by allowing real-time feedback and tracking changes. It also generates comprehensive documentation for developers, ensuring a smooth transition from design to implementation while minimizing miscommunication risks. By identifying design issues early, Axure helps reduce costly revisions during development, ultimately accelerating the time-to-market for products (Axure, 2025).

2.3.3 JAD (Joint Application Development)

JAD is a structured process where users, managers, and systems analysts collaborate in a series of intensive meetings over a few days to review system requirements. This joint effort streamlines the requirements review process and enhances its efficiency (Valacich *et al.,* 2015: 165). Figure 6 illustrates the typical layout of a JAD session room.

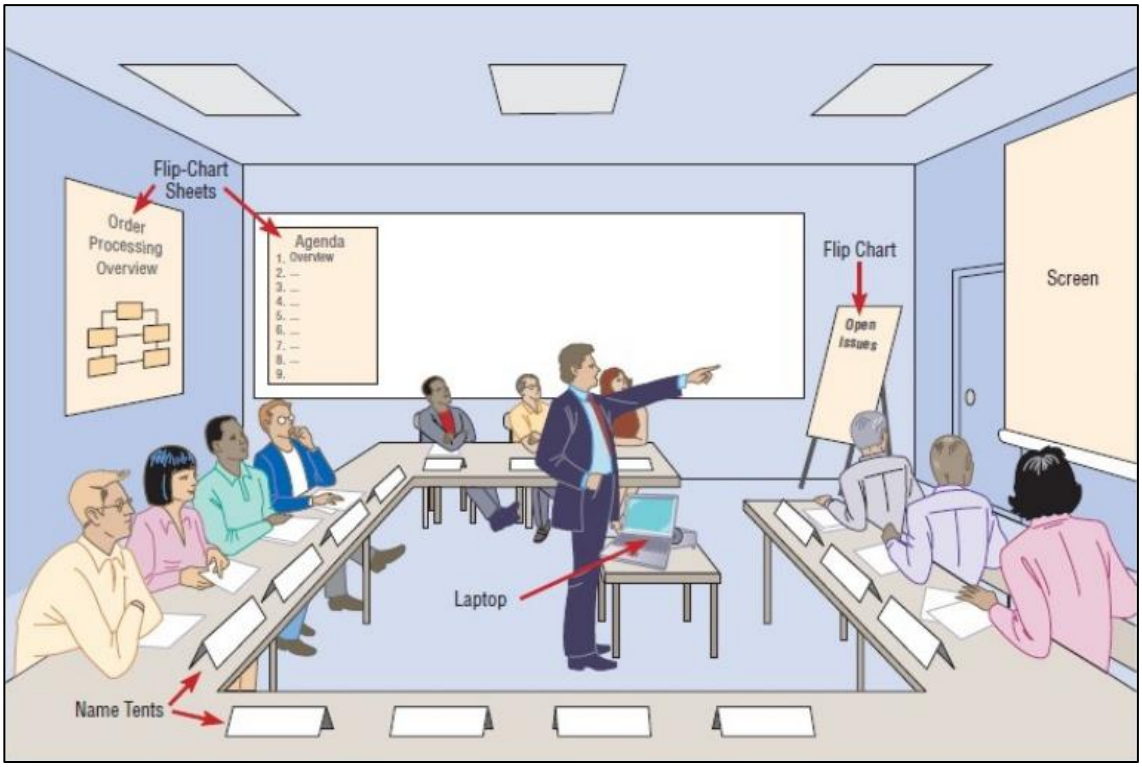

Figure 6 – JAD session: typical room layout (Valacich, *et al.*, 2015:166)

2.3.4 RAD (Rapid Application Development)

RAD emerged as a response to the structured methodologies developed in the 1970s and 1980s, aiming to deliver software rapidly while adapting to changing requirements. RAD emphasizes continuous user involvement through prototyping, which allows for gathering user feedback throughout the development process. By engaging users early, RAD increases the likelihood of meeting customer requirements (Valacich *et al.,* 2015:46). Figure 7 compares RAD to the standard SDLC.

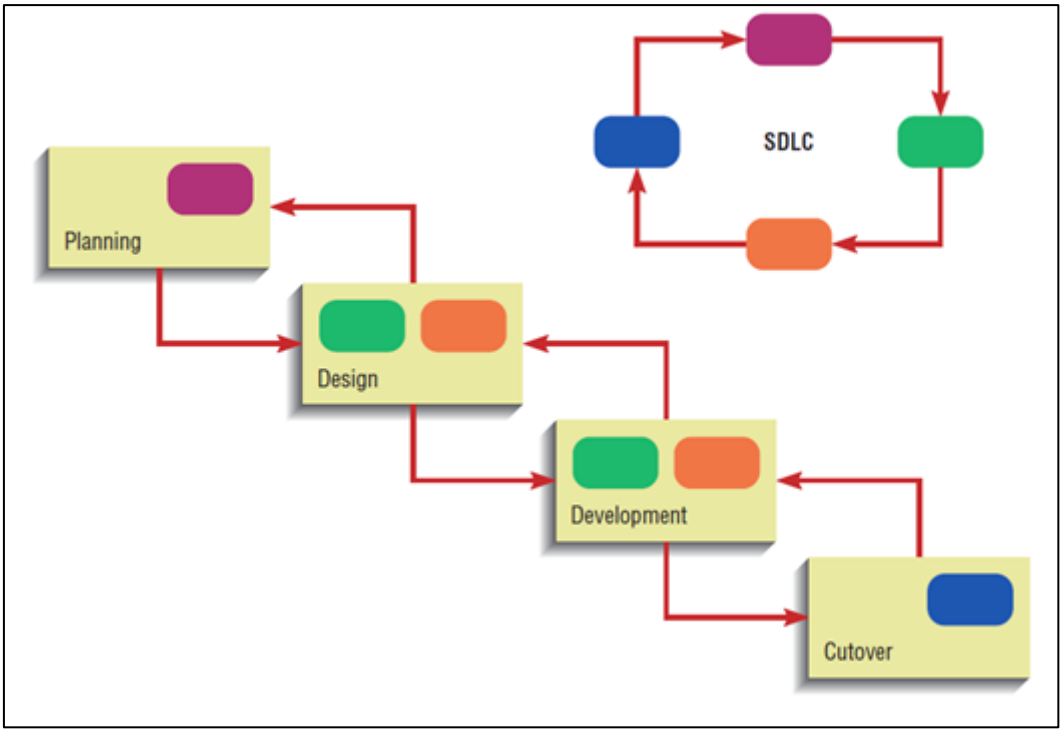

Figure 7 – RAD compared to standard SDLC (Valacich, *et al.*, 2015:46)

The RAD phases are iterative and repeated many times by the development team until the end- product satisfies the user requirements. Figure 8 shows the phases of RAD (Lomio & Mejidana, 2023).

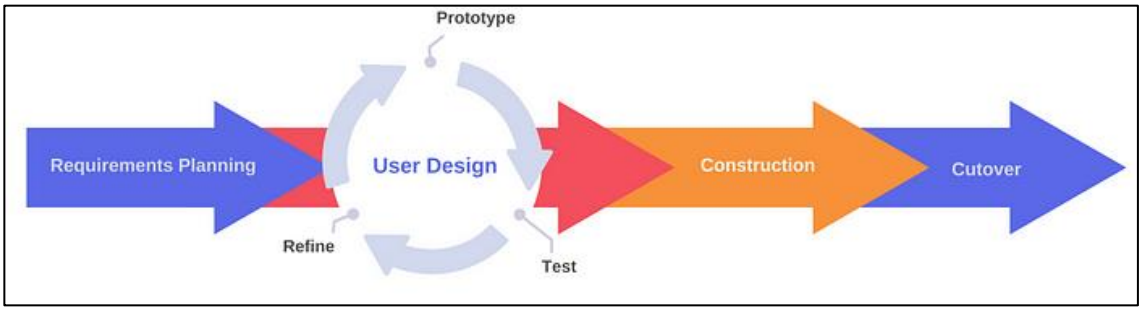

Figure 8 – The phases of RAD (Lomio & Mejidana, 2023)

Barry Boehm, a software engineer, developed the first RAD alternative, known as the "spiral model" in 1986 (Icasas, 2024). The spiral model emphasizes the importance of risk analysis in every phase of systems development. Risk is evaluated and addressed each time the process reaches the risk analysis step. Figure 9 depicts the spiral model.

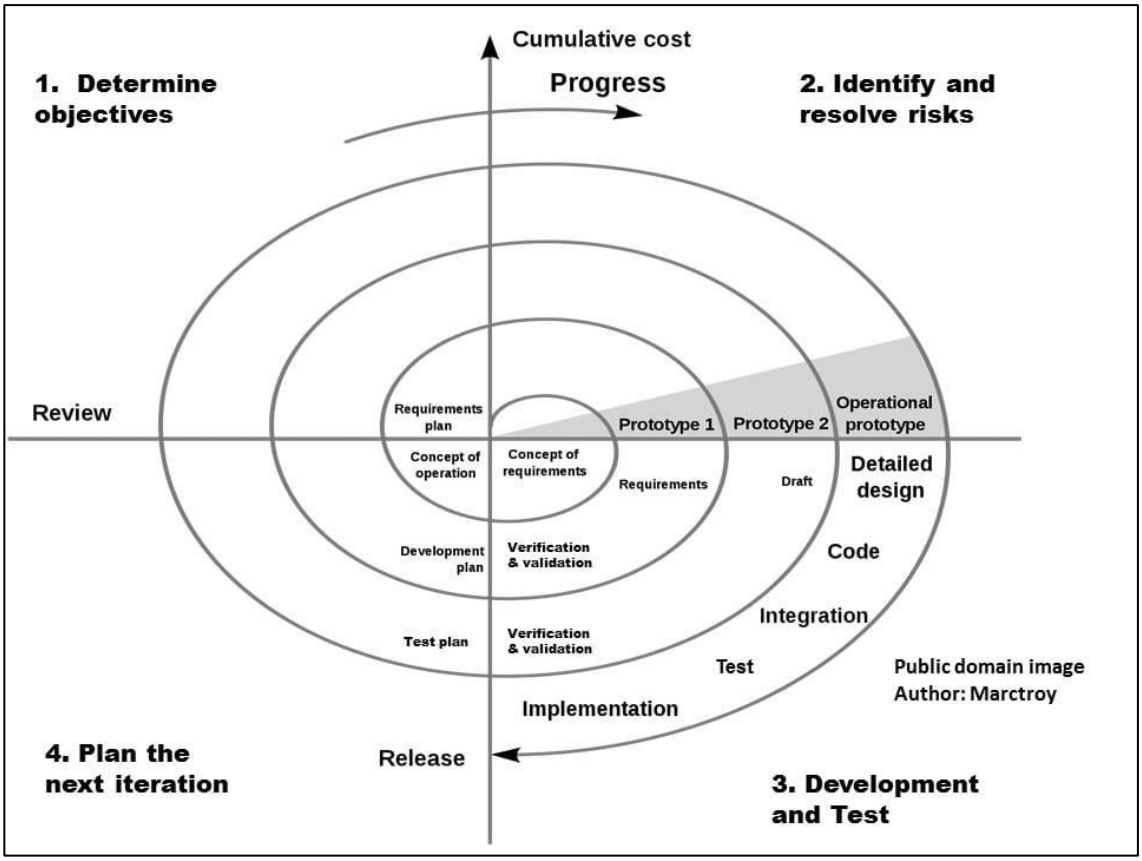

Figure 9 – The Spiral model (Victor, 2025)

2.3.5 eXtreme programming (XP)

eXtreme Programming (XP), developed by Kent Beck, is a prominent agile methodology recognized for enhancing software quality and accommodating changing requirements. The term "extreme" reflects the methodology's emphasis on pushing iterative development to its limits. With XP, teams aim to deliver functional software versions frequently, sometimes multiple times a day (Raeburn, 2025).

In XP, increments of software are delivered to customers bi-weekly, with testing conducted after each release (Raeburn, 2025). Upon customer acceptance and successful testing, team members finalize the version. This methodology is best suited for experienced developers who possess a strong understanding of the domain and system type. Additionally, it requires full-time customer involvement, treating them as integral team members.

A key practice in XP is pair programming, where two developers collaborate closely, reviewing each other's code and occasionally switching roles. This approach fosters collective ownership of the code and ensures knowledge sharing

among team members. XP encourages continuous refactoring of code until both the customer and the team are satisfied with the product iterations (Raeburn, 2025). The development process follows a cyclical pattern: work is divided into increments that are planned, developed, tested, and released for evaluation, repeating until the final product is complete.

XP has its drawbacks, including the potential for scope creep—where project scope expands uncontrollably without managing cost, time, and quality constraints. It can also be ineffective if not implemented by a skilled team and may lack the structure and documentation typical of traditional methodologies. Furthermore, systems design may not receive as much attention as in conventional approaches, and the frequent meetings required between teams and customers can incur additional costs for clients.

There are many other agile methodologies, such as:

- Kanban
- Crystal
- Dynamic Systems Development Method (DSDM)
- SCRUM

Activity

Study the following link: https://www.scrum.org/ Conduct research on SCRUM (an agile methodology), and describe the SCRUM terms below:
-Scrum master
-Daily scrum meeting
-Product backlog
-Sprint backlog
-Burndown chart
Time allocation: 20 minutes

2.4 SOFTWARE DEVELOPMENT PLATFORMS AND TOOLS

A software platform serves as a foundational environment that enables developers and users to create, run, and manage applications. A software tool is a specific application or program designed to perform a particular task.

System professionals play an important role in the software engineering process by ensuring a strong understanding of methodologies, techniques, and tools. By effectively applying these elements, they enhance the likelihood of developing an effective and reliable information system (Valacich *et al.,* 2015:31).

A technique refers to a particular method for performing a task to reach a specific goal. A tool is an instrument or computer program utilised to implement the technique. A methodology serves as a structured framework that encompasses various stages, guiding the process toward achieving a comprehensive outcome (Valacich *et al.,* 2015:31). Figure 10 shows the software engineering process.

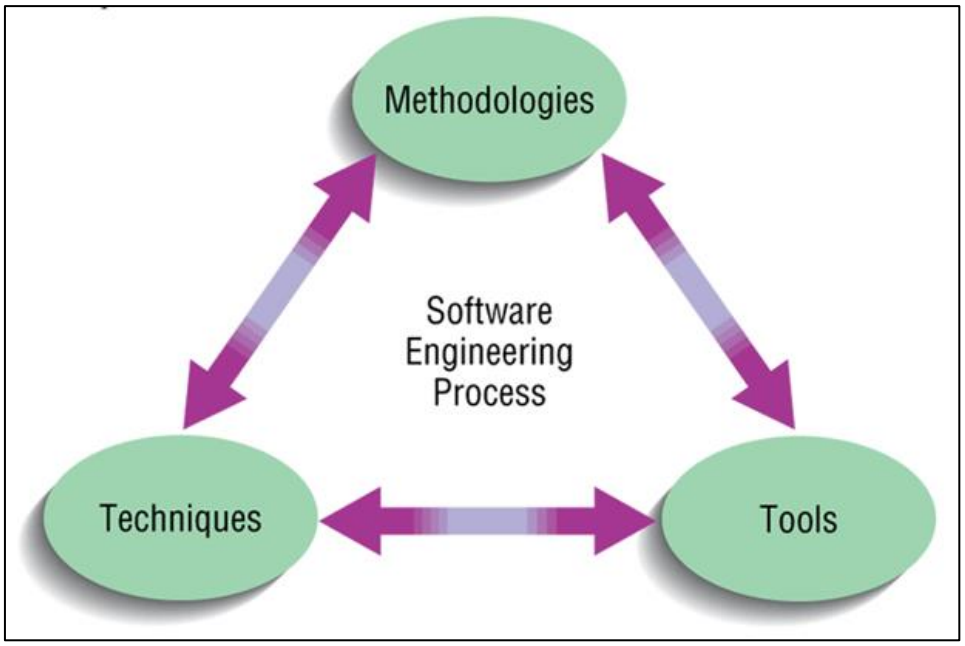

Figure 10 – The software engineering process (Valacich *et al.,* 2015:31).

The following example shows the link between the three concepts. In the construction industry, designing the foundation of a new building requires a specific technique. Architects and designers utilize specialised software as the tool to facilitate this design process. Additionally, the team will implement a selected methodology or framework commonly used in foundation engineering to ensure a systematic approach toward achieving a successful outcome.

There are various software platforms and tools available in the field of software development. The following top platforms and tools are discussed by Cardozo (2024) and Michalowski (2025):

Integrated Development Environments (IDEs) and Code Editors:

- Visual Studio Code (VS Code): A widely used, flexible, and free IDE that supports multiple languages and platforms. It has a fast source code editor, perfect for day-to-day use.
- NetBeans: A free, open-source IDE used for developing desktop, mobile, and web applications.

Version Control Systems:

- GitHub: A leading web-based platform for hosting code repositories, enabling developers to collaborate, manage projects, and review code. It uses Git - a version control system.
- Bitbucket: Atlassian owns Bitbucket. It is a Git-based source code repository hosting service for software developers.

Project Management and Collaboration Tools:

- Jira: A project management tool to track issues, automate workflows and manage general project management tasks.
- Slack: A cloud-based, AI-powered messaging platform for project teams to support team communication and collaboration.

Cloud Platforms:

- Amazon Web Services (AWS): A cloud platform providing IT resources on demand and on a pay-as-you-go basis. It offers computing, storage and database capabilities.
- Microsoft Azure: A cloud platform offering computational, networking, storage, analytics and AI capabilities, for organisations to manage a range of applications.

There are more categories of software tools for software development. These include testing and quality assurance tools, database management tools, containerisation and orchestration tools, and build automation tools (Michalowski, 2025).

Computer Aided Software Engineering (CASE) tools are automated software tools that systems professionals use to develop information systems. According to Valacich *et al.* (2015:45), there are many general CASE tool types for different development stages.

- Diagramming tools
- Analysis tools
- Computer display and report generators
- Repository
- Code generators
- Documentation generators

Diagramming tools are essential for systems analysts to create various diagram types, such as Use Case diagrams, Entity Relationship Diagrams (ERDs), Data Flow Diagrams (DFDs), and many others. Popular diagramming tools include IBM System Architect, Lucidchart, Miro, Microsoft Visio, and SmartDraw. Figure 11 shows a screenshot of the SmartDraw official website and what the company offers for software engineering. The mobile versions of these tools improve efficiency of mobile teams. There are also open-source modelling tools available. The team should assess the availability and functionality of a tool before using it.

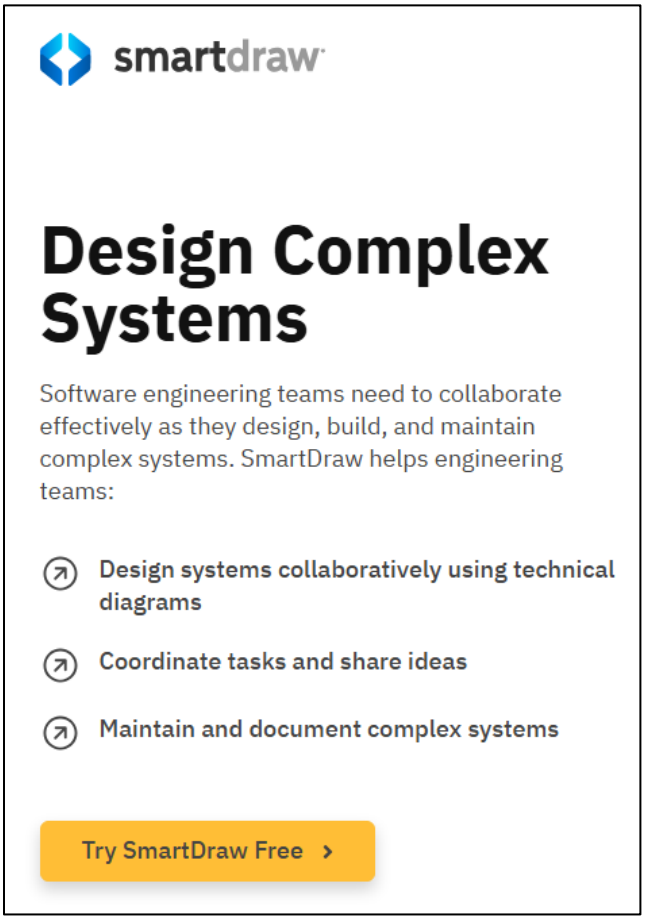

Figure 11 – SmartDraw for software engineering (Smartdraw, 2025).

Software diagrams are essential for improving understanding, communication, and efficiency throughout software development. Software teams should use well-designed tools to support their diagramming needs.

Activity

Study the functionality of LucidChart (https://www.lucidchart.com/pages).
Time allocation: 10 minutes

Levrel (2025) shows a list of Artificial Intelligence (AI)-powered software development assistants supporting code generation:

- Qodo Gen
- Cursor AI
- Codeium
- AskCodi

These are only a few examples of AI code generators on the market. Most of these tools support software scalability, maintenance and smooth integration processes. It also supports efficient code writing and error resolution (Levrel, 2025).

Activity

Study the following link: https://whatfix.com/blog/software-documentation-tools/. Conduct research on Software documentation tools.
Time allocation: 10 minutes

Summary

Software development teams use specific methodologies for systems development based on the type of project and organisation. Sequential development is a methodology where each phase flows into the next, and the subsequent phase cannot begin until the previous one is fully completed. The waterfall model is a sequential method.

Iterative incremental development is a methodology where systems are delivered in increments. This approach is closely associated with agile development, which focuses on delivering working components of software in small, frequent increments to improve and refine the system.

There is a strong movement towards agile development in the field of IT, specifically for the development of AI applications. Software developers should be competent in using agile methods like SCRUM in today's industry.

There are various platforms and tools available for software development. Organisations should conduct research and decide which platforms and tools suit their organisation's needs the best.

Self-Assessment Questions

1. Describe the sequential nature of the Waterfall model for systems development.

2. Discuss why agile methodologies are gaining attention in the field of software development.

3. Explain the importance of the daily SCRUM meeting used by development teams.

4. Discuss some of the best software tools on the market for project management.

TOPIC 3
Software development methods, tools, and modelling techniques

3. INTRODUCTION

This topic relates to the following module outcome:

3. Apply methods, tools and modelling techniques commonly employed during the various phases of software development.
4. Demonstrate an understanding of the importance of software maintenance and documentation.
5. Identify modern software development and management platforms, tools, and services, and outline the nature of the support provided.

In this topic, you will gain knowledge in the following areas:

3.1 Software development methods and tools

3.2 Software development modelling techniques

3.3 Software modelling examples

3.4 Software modelling exercises

Prescribed reading

Letaw, 2024:

- Requirements (Page 30)
- How Diagrams Help (Page 46)
- What Is UML? (Page 47)
- Class Diagrams (Page 49)

3.1 SOFTWARE DEVELOPMENT METHODS AND TOOLS

Software development teams use a variety of methods or techniques to successfully complete software development projects. Whether a team adopts a sequential approach such as the Waterfall model or a more flexible agile methodology like Scrum or Extreme Programming, there are numerous techniques and tools available to support developers throughout the various phases of software development.

During the first phases, fact-finding techniques and tools are used to determine the system requirements. During all phases, team collaboration tools are used for communication. For example, team meetings occur throughout a software development project. The team meeting would be the technique, and the tools could be Microsoft Teams to create a space for the online meeting, Microsoft Word to capture the meeting minutes and Microsoft Outlook to distribute the communication content.

During the analysis phase, analysts use tools to draw many types of diagrams to model the system requirements. Diagrams can be beneficial to developers in at least two ways. Firstly, they assist in the planning of software development. By creating diagrams during the planning phase, you can effectively convey to the development team what needs to be built and assess whether your design is solid, for example, whether it's clear, logically structured, and aligned with the project's quality goals. Secondly, they help illustrate software that has already been developed. In this context, diagrams are useful for documentation purposes and for assessing how well the final product meets expectations. Including diagrams in documentation explains aspects of your software to others. This could involve a variety of audiences, depending on who needs to understand the system (Letaw, 2024).

During the coding or implementation phase, developers use Integrated Development Environments (IDEs) that provide code editing, debugging, and compiling tools. They use compilers and interpreters to translate source code into machine code or execute it line-by-line. Other tools are debugging tools, version control system tools, build automation tools and testing tools.

Using software tools throughout the Software Development Life Cycle (SDLC) is vital for ensuring the successful planning, development, deployment, and maintenance of software systems. These tools support each phase of the SDLC, from requirements gathering to design, coding, testing, deployment, and maintenance, by enhancing accuracy, efficiency, and collaboration.

Activity

Study the following link: https://www.guru99.com/testing-tools.html. Conduct more research on tools that could be used during the testing phase of software development.
Time allocation: 15 minutes

3.2 SOFTWARE DEVELOPMENT MODELLING TECHNIQUES

3.2.1 Modelling techniques overview

In the Information Technology (IT) industry, systems analysts create both logical and physical models, often in the form of diagrams before moving into the development phase. These diagrams serve as blueprints that guide programmers in building the system based on the analysts' detailed specifications. Creating diagrams allows analysts to examine the proposed system from multiple perspectives and concentrate on different components. To aid in this process, analysts often use modelling tools, known as CASE (Computer-Aided Software Engineering) tools, which help ensure the diagrams are complete and consistent. Different types of diagrams are used to highlight various aspects of the system.

For process modelling, analysts draw flow charts and data flow diagrams (DFDs). The context diagram is the highest-level DFD, offering a broad overview of the entire system and its interactions with external entities. From this, lower-level DFDs are developed to explore each process in greater detail. A Data Flow Diagram (DFD) is an essential tool for modelling business and system processes. It illustrates the flow of data through a system, including inputs, outputs, and how data is processed (Whitten & Bentley, 2007:162). Often referred to as a "process model," a DFD specifically represents the various processes within a system. Systems analysts typically create DFDs at multiple levels of detail, breaking down complex systems into smaller, more manageable sub-processes. This breakdown starts with a decomposition diagram, which is then used to develop DFDs and other supporting diagrams.

Conceptual data modelling focuses on representing an organisation’s data, and the aim is to highlight the rules related to the meaning and interrelationships among an organisation’s data. The main aim of conceptual data modelling is to

create an Entity-Relationship Diagram (E-R diagram or ERD).

Unified Modelling Language (UML) is a collection of visual notations used to describe and design software systems through diagrams. While it is particularly suited for object-oriented software, several elements of UML can be applied to other software types as well. Various UML notations are tailored for specific diagram types, each serving a unique role in modelling. Originally introduced in 1994, UML was standardized by the Object Management Group (OMG) in 1997 and later adopted as an ISO standard in 2005 (Letaw, 2024).

According to the Object Management Group, the current version of UML is 2.5.1. The document is available on the module Canvas page. OMG's mission statement is: "To generate technology standards (250+) that provide quantifiable real-world value to all vertical industries. That is why we are dedicated to bringing together our international membership community of end-users, researchers and vendors in academia government and industry to develop and revise our standards as technologies change over time" (OMG, 2025).

Analysts do not have the time (or project money) to draw every single diagram. The team usually decides which diagrams would add the most value for further software development, based on the type of methodology followed and the nature of the project.

The following list of UML 2.0 Diagrams is adapted from Whitten & Bentley, 2007:382)**:**

- Use case diagram: Depicts the interaction between the system and external systems and users. It graphically describes who will use the system and how the user expects to interact with it. A use-case narrative also textually describes the sequence of steps for each interaction.
- Activity diagram: Shows the sequential flow of activities in a use-case or business process. It can also model internal logic within the system.
- Class diagram: Represents the system's object structure, showing object classes and the relationships between them.
- Object diagram: Like a class diagram, but models actual object instances with current attribute values, providing a snapshot at a specific moment.
- State machine diagram: Illustrates how events cause changes in an object's state over its lifetime, showing various states and transitions.
- Composite structure diagram: Breaks down the internal structure of a class, component, or use case.
- Sequence diagram: Shows how objects interact through message exchanges during a use case or operation, including the sequence of these messages.

- Communication diagram: Formerly called a collaboration diagram in UML 1.X, it focuses on the structure of object interactions via messages, emphasizing organization rather than timing.
- Interaction overview diagram: Combines sequence and activity diagram features to show object interactions within each activity of a use case.
- Timing diagram: Focuses on timing constraints and state changes of objects, particularly useful in embedded software design.
- Component diagram: Illustrates the organization of programming code into components and how these components interact.
- Deployment diagram: Displays how software components are configured on the physical system hardware nodes.
- Package diagram: Shows how UML elements (like classes) are grouped into packages and how those packages depend on each other.

3.2.2 Use case modelling

Use cases provide a structured and formalised approach to specifying functional requirements, detailing the expected behaviour of a system during user interactions (Letaw, 2024).

Use case modelling, a key aspect of object-oriented modelling, has also been adopted in traditional development environments due to its effectiveness in capturing system requirements. It represents the functions a system should perform from the perspectives of users and stakeholders, making it a widely accepted method for defining, documenting, and understanding functional requirements (Whitten & Bentley, 2007). Figure 12 shows the interaction between an information system and actors.

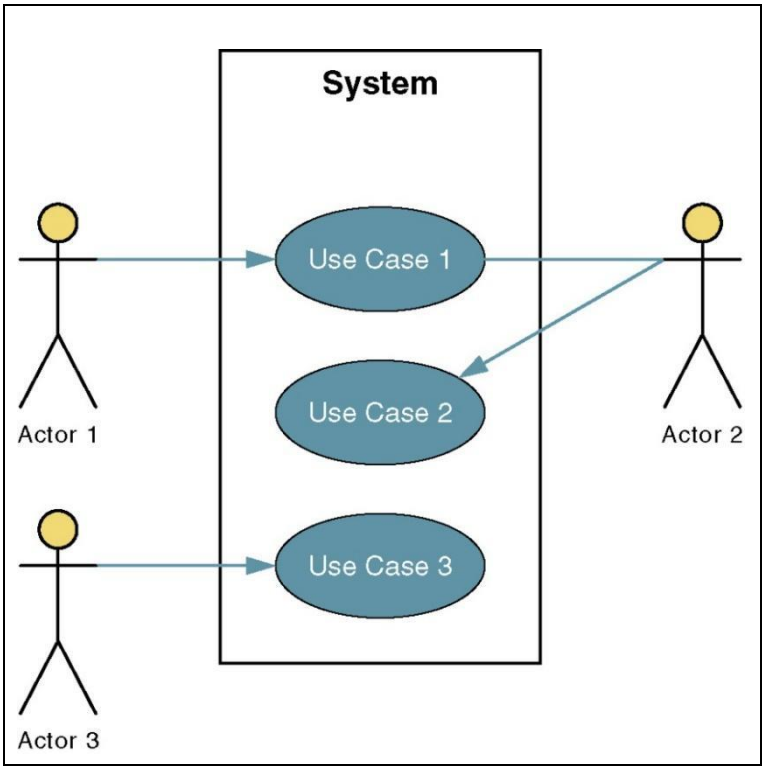

Figure 12 – Sample use case diagram (Whitten & Bentley, 2007)

Before developing a use case model, it is important to first identify the actors, such as users, external systems, or time-based triggers, as well as the potential use case events and the relationships among them. Each use case event should be action-oriented and begin with a verb (e.g., 'Generate monthly student attendance report'). A complete use case diagram provides a visual representation of all possible interactions within the system.

Use cases are initially defined at a high level during the requirements phase and should be continuously revised and refined throughout the system's life cycle as further analysis and design activities are conducted.

The following are use case modelling steps by Whitten & Bentley (2007):

1. Identify the actors.
2. Identify the use cases.
3. Construct use-case model diagram.
4. Document the requirements use-case narratives.

According to Letaw (2024), every use case includes the following:

Name: A concise title for the use case, typically beginning with a verb (e.g., "Schedule weekly wellness check"). It briefly states the user's objective that the use case will describe.

Actor(s): The individual(s), system(s), or device(s) interacting with the software—these may be human, nonhuman, or automated entities (e.g., "Medical staff").

Flow of Events: A sequential description of the interactions between the actor and the system. This outlines the typical steps taken to achieve the use case goal, also known as the basic course of action or success scenario.

An actor is any entity that interacts with a system to exchange information. This may include a person, an organization, another information system, an external device, or even a temporal element such as time. A temporal event is a specific type of system event triggered by time itself, in which time acts as the actor. Actors should be named using a noun or noun phrase. To identify actors, consider the following questions (Whitten & Bentley, 2007):

- Who or what provides input to the system?
- Who or what receives output from the system?
- Are interfaces required with other systems?
- Are there events automatically triggered at predetermined times?
- Who will maintain the information within the system?"

A **primary business actor** is the stakeholder who primarily benefits from the execution of a use case. For example, an employee receiving a paycheck is considered a primary business actor.

A **primary system actor** is the stakeholder who directly interacts with the system to initiate or trigger a business or system event, such as a bank teller entering deposit information.

An **external server actor** is a stakeholder that responds to a request from the use case, like a credit bureau authorizing a credit card charge.

An **external receiver actor** is a stakeholder who is not the primary actor but still receives something of value from the use case, such as a warehouse receiving a packing slip (Whitten & Bentley, 2007).

Use cases should be named using a verb phrase that clearly defines the actor's objective (e.g. 'Place Subscription Order'). To identify use cases, ask these questions (Whitten & Bentley, 2007):

- What are the primary responsibilities of the actor?
- What data does the actor require from the system?
- What data does the actor input into the system?
- Should the system notify the actor about any updates or events?

- Should the actor notify the system about any updates or events?

An association is a relationship (interaction) between an actor and a use case and represented as a solid line connecting an actor and a use case. If the association includes an arrowhead pointing to the use case (Figure 13), it signifies that the actor initiated the use case (1). If there is no arrowhead, it indicates a receiver actor (2). Associations can be either bidirectional or unidirectional, depending on the nature of the interaction between the actor and the use case (Whitten & Bentley, 2007).

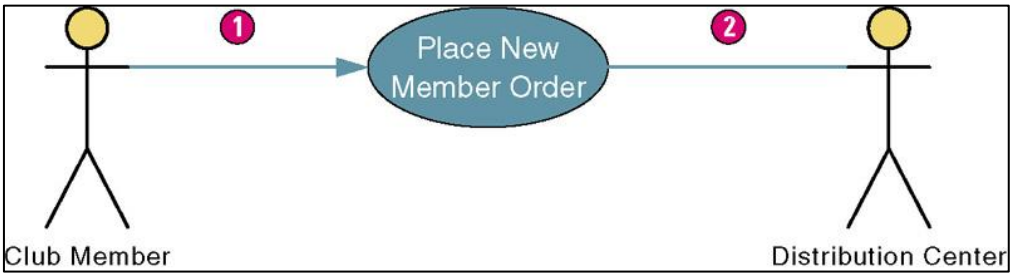

Figure 13 – Use case diagram association (Whitten & Bentley, 2007)

3.2.3 Class diagram

A class diagram provides a visual representation of a system's classes and the static relationships among them. It also illustrates the properties and operations associated with each class. Properties define the structure of a class, such as its instance variables, while operations specify the behaviour or functionality that the class offers (Letaw, 2024).

Figure 14 shows a class diagram example. There are relationships in the diagram between three classes: Customer, Order, and SharedOrder. An Order has one Customer; but the same Customer can be linked to many Orders. A SharedOrder is a type of Order that can consist of many Customers. The classes have attributes (id) and operations (getId()).

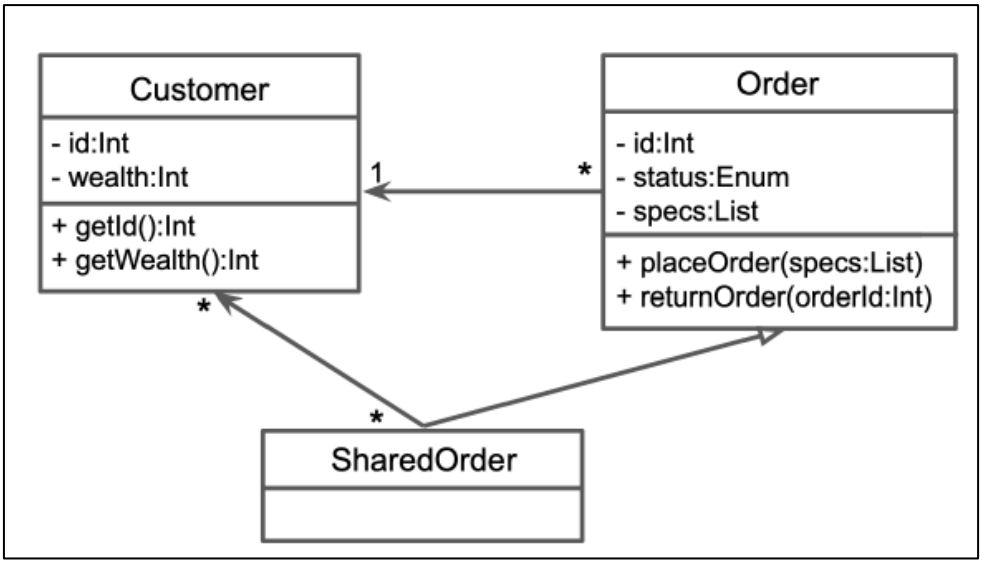

Figure 14 – Class diagram example (Letaw, 2024)

Figure 15 shows an example of a class. Attributes are the properties listed within a class box, while operations correspond to the methods of the class. Visibility indicators are used to denote access levels: '+' signifies a public method, '–' indicates a private method, and '#' represents a protected method. The notation also includes attribute data types (e.g., *int*, *Token*), method parameters and return types, as well as default values assigned to attributes.

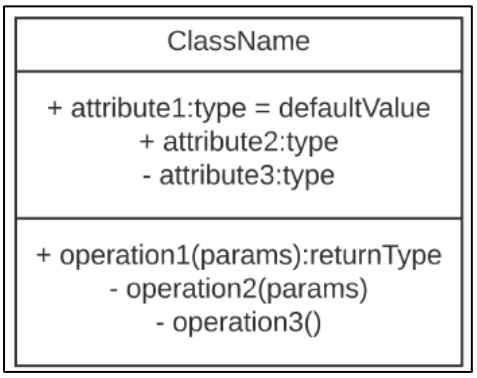

Figure 15 – Class diagram example (Letaw, 2024)

An association represents a relationship in which one class holds a reference to one or more objects of another class, typically in the form of a property (Figure 16). The direction of the reference may not be explicitly defined, for example, it may be unclear whether *Class1* references *Class2*, *Class2* references *Class1*, or if the association is bidirectional.

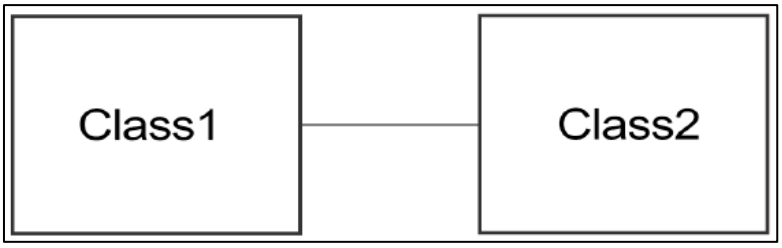

Figure 16 – Class association (Letaw, 2024)

The arrow in Figure 17 shows that Class1 has a Class2.

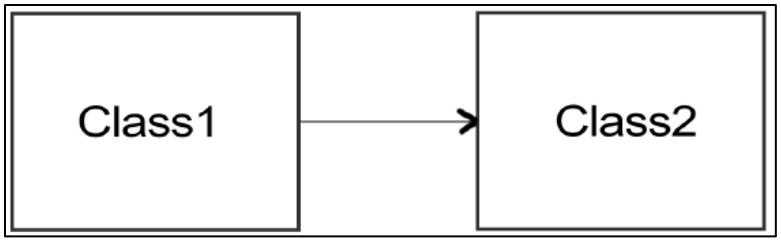

Figure 17 – Unidirectional association (Letaw, 2024)

Multiplicity defines how many instances of one class can be associated with a single instance of another class. It is shown at the ends of association lines in UML class diagrams and helps describe the cardinality of the relationship. Figure 18 shows that Zero or more instances of Class1 have properties named instance1 containing zero or one instance of Class2.

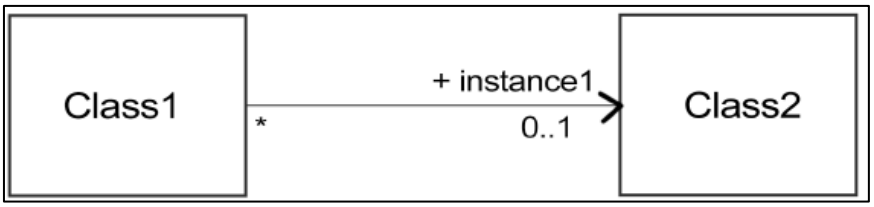

Figure 18 – Unidirectional association with property name, target multiplicity, and source multiplicity (Letaw, 2024)

Figure 19 shows that Class1 has zero or more instances of Class2. Class2 has exactly one instance of Class1.

Figure 19 – Bidirectional association and target/source multiplicity (Letaw, 2024)

Inheritance in a class diagram represents a hierarchical relationship between classes, where one class (the *subclass* or *child class*) inherits attributes and operations from another class (the *superclass* or *parent class*). This allows for code reuse and logical organization of shared behaviour. Figure 20 shows that Class2 is a subclass of Class1; in other words, Class2 is a type of Class1.

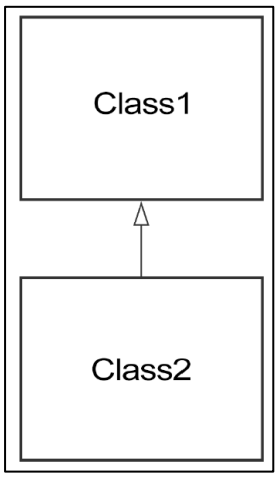

Figure 20 – Class inheritance (Letaw, 2024)

3.2.4 Activity diagram

The purpose of an Activity diagram is to model the dynamic aspects of a system, with a focus on the flow of control from one activity to another. It is used to describe the logic of business processes or software system processes, helping to visualize how tasks are carried out in sequence or in parallel. Additionally, activity diagrams are valuable for modelling workflows between different components of a system and for representing concurrent (parallel) operations within a process. Figure 21 shows the notation used on an activity diagram.

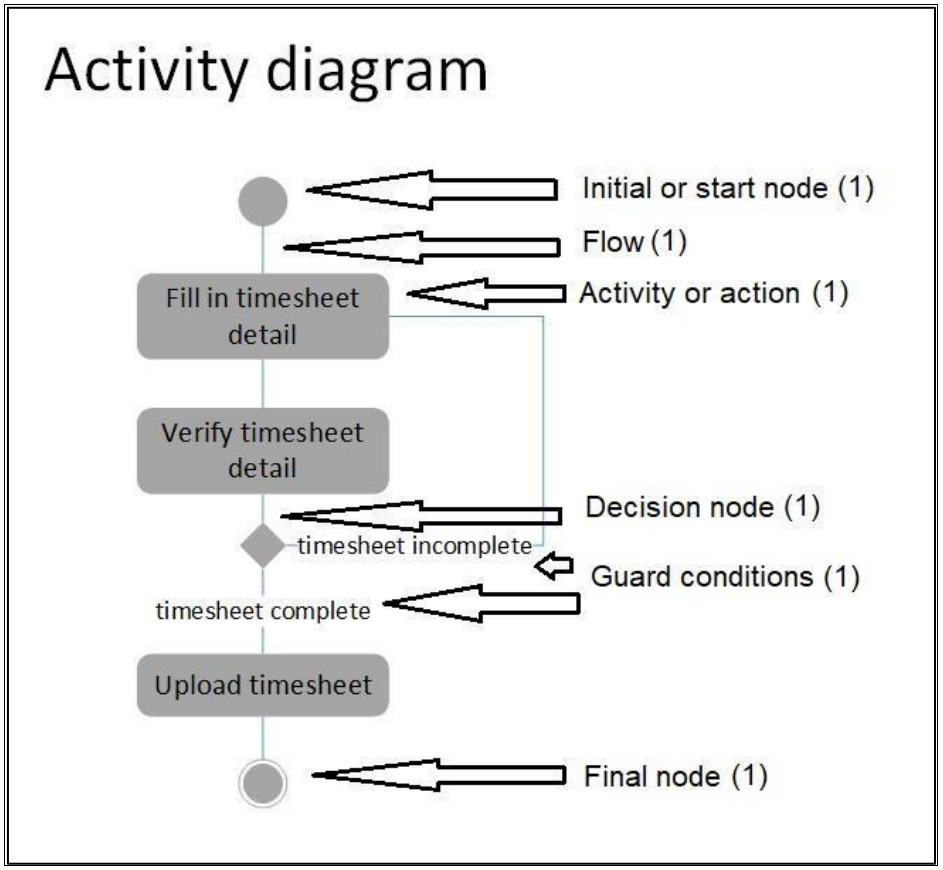

Figure 21 – Activity diagram notation

The following plan could be used before drawing an activity diagram for the scenario of buying a new house:

- Start Node (●)
- Search for House (Activity)
- Shortlist Houses (Activity)
- Visit Houses (Activity)
- Decision: Found Suitable House? (Diamond)
  - Yes → Proceed to Next Step
  - No → Go back to Search for House
- Get Pre-Approval for Mortgage (Activity)
- Make an Offer (Activity)
- Decision: Offer Accepted? (Diamond)
  - Yes → Proceed to Next Step
  - No → Return to Shortlist Houses
- Home Inspection (Activity)
- Decision: Satisfied with Inspection? (Diamond)
  - Yes → Proceed
  - No → Return to Shortlist Houses or Renegotiate

- Finalize Mortgage (Activity)
- Sign Contract and Close Deal (Activity)
- Move In (Activity)
- End Node (◎)

Figure 22 shows the corresponding activity diagram for the plan outlined above.

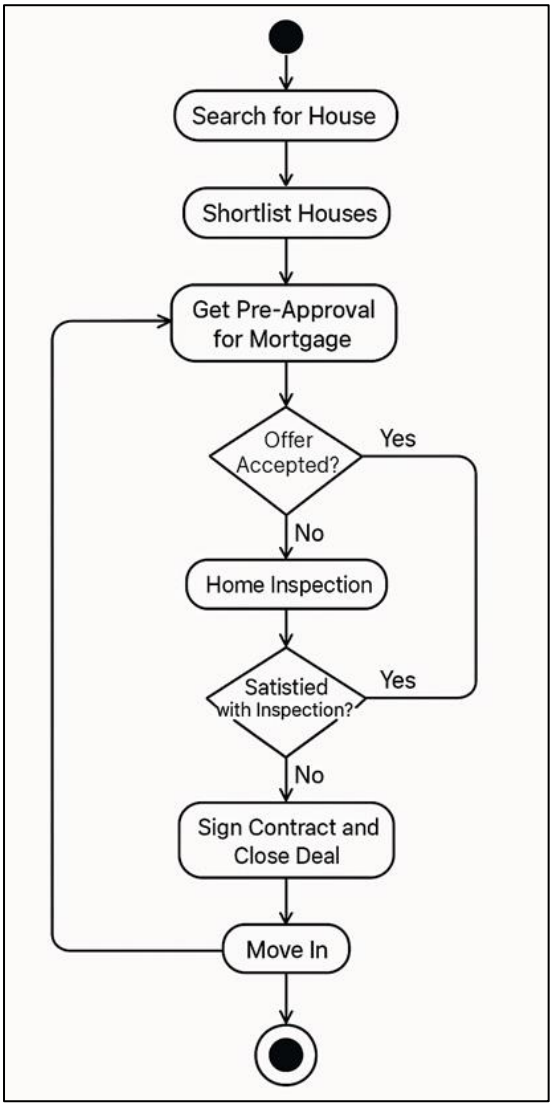

Figure 22 – Activity diagram: Buying a new house

3.3 SOFTWARE MODELLING EXAMPLES

3.3.1 Use case diagram example

Case study: PetStore Waiting list system

A Pet store in Gauteng approached you to develop a system to support waiting lists at the shop. Many customers place request for certain animals and breeds and the staff usually captures this manually in a book. The processes of following up with customers on pet availability and with breeders who advertise pets at the store, requires improvement.

You conducted an interview with the manager and staff, and it was determined that the new waiting list system should support the following functions: Staff should be able to add new customers on the system who want to be on the waiting list for possible animals. Staff also need to update customer details should their details change. Should a customer receive the animal, the customer could request to be removed from the waiting list.

Customers should be able to contact the store anytime to submit a new requirement for a certain animal. Each submission is automatically forwarded to the matching breeder of that animal type in the form of an email.

Breeders should be able to contact the store for staff to add which animals are available for that specific breeder. Should there be a match between the animals submitted and the customer request, the system should send an automatic notification to the customer via SMS to inform the customer of a possible availability of required animal.

The owner who is also the store manager, has requested that it could be useful if the system generates an animal list at any time to see which animals are available. The manager also wants to see a list of customers waiting to find available animals at any given time. The manager and staff indicated that the system should be user-friendly and incorporate the colours of the store's current branding.

Figure 23 shows a use diagram based on the case study.

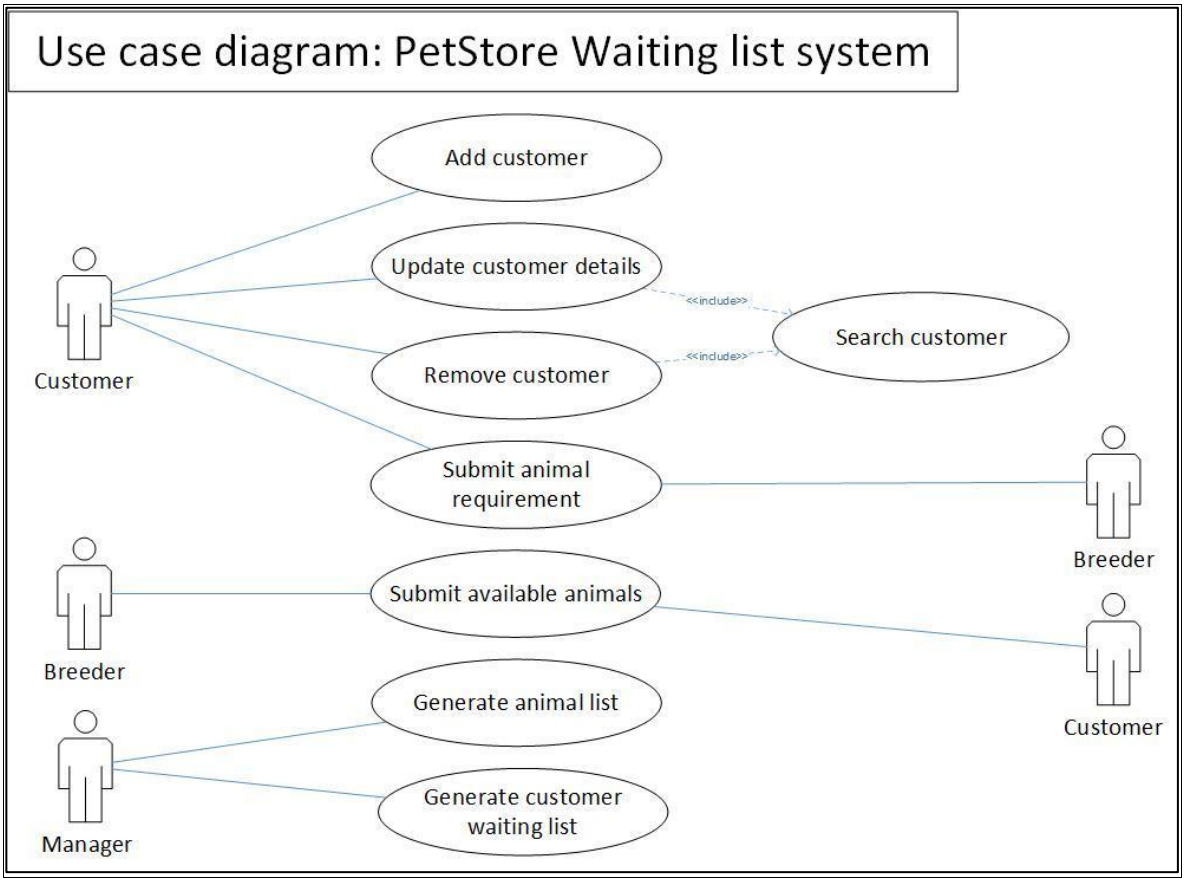

Figure 23 – Use case diagram: PetStore Waiting list system

3.3.2 Class diagram examples

a) Case study: Sunrise General Hospital

Sunrise General Hospital is implementing a digital Patient Management System to streamline patient admissions, treatment, and bed assignments. The system must manage different types of patients, track bed availability, and record interactions between patients and physicians.

- Patient – A general term for any individual receiving medical care at the hospital. Every patient is registered in the hospital system.
- Outpatient – A type of patient who visits the hospital for a consultation, diagnosis, or minor treatment without being admitted.
- Resident Patient – A patient who is admitted to the hospital for overnight or longer stays. These patients require bed allocation and continuous monitoring.
- Physician – A medical professional responsible for diagnosing and treating patients. Physicians can be assigned to both outpatients and resident patients.

- Bed – A physical resource that is assigned to resident patients. Bed availability must be tracked to avoid overbooking.

Figure 24 shows the class diagram created using the information.

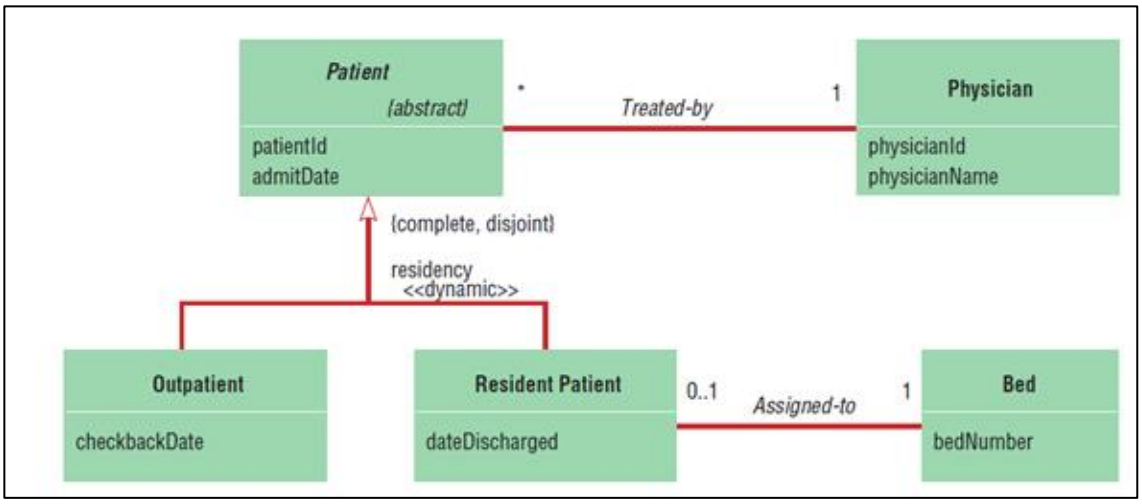

Figure 24 – Class diagram: Sunrise General Hospital

b) Case study: Higher education institution

A global higher education institution employs 1800 academic staff members from many countries. The institution currently employs 1100 part-time lecturers, of which some are appointed on-site (on campuses) and others are employed as online lecturers who work remotely. The institution employees 600 full-time lecturers globally. The institution requires a new information system to manage timesheets and payslips of part-time lecturers.

Figure 25 shows a class diagram for the higher education institution to clearly show generalisation and specialisation associations (no attributes or operations required).

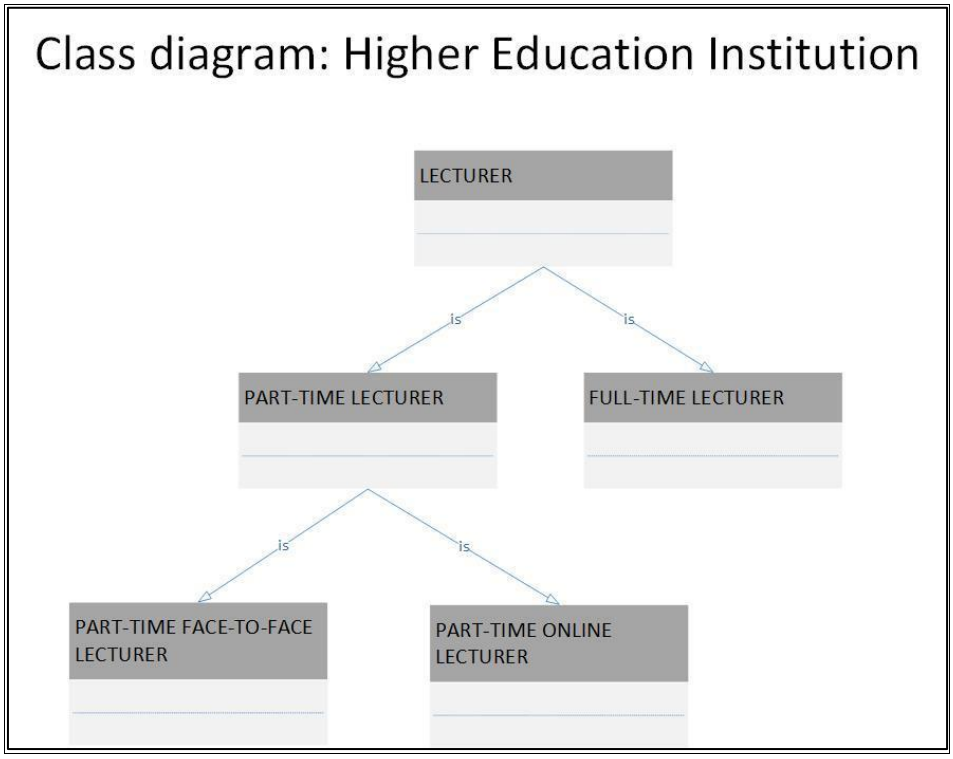

Figure 25 – Class diagram: Higher Education Institution

3.3.3 Activity diagram examples

a) Figure 26 shows an activity diagram for the scenario of eating dinner:

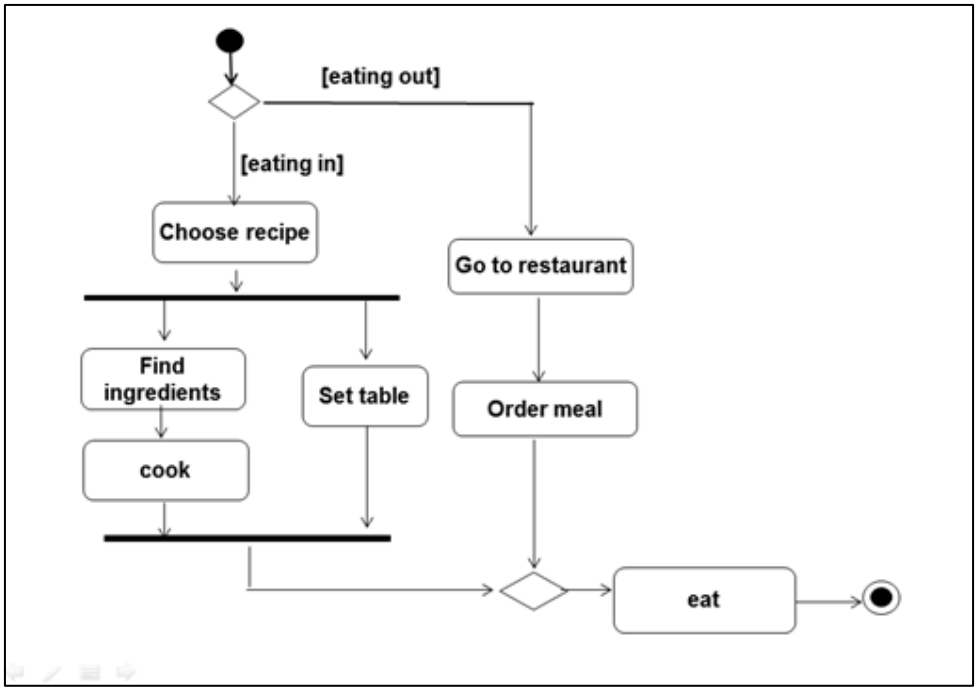

Figure 26 – Activity diagram: Eating dinner

b) Figure 27 shows an activity diagram for the business process of a lecturer submitting a timesheet on a timesheet system. A few assumptions were made where necessary.

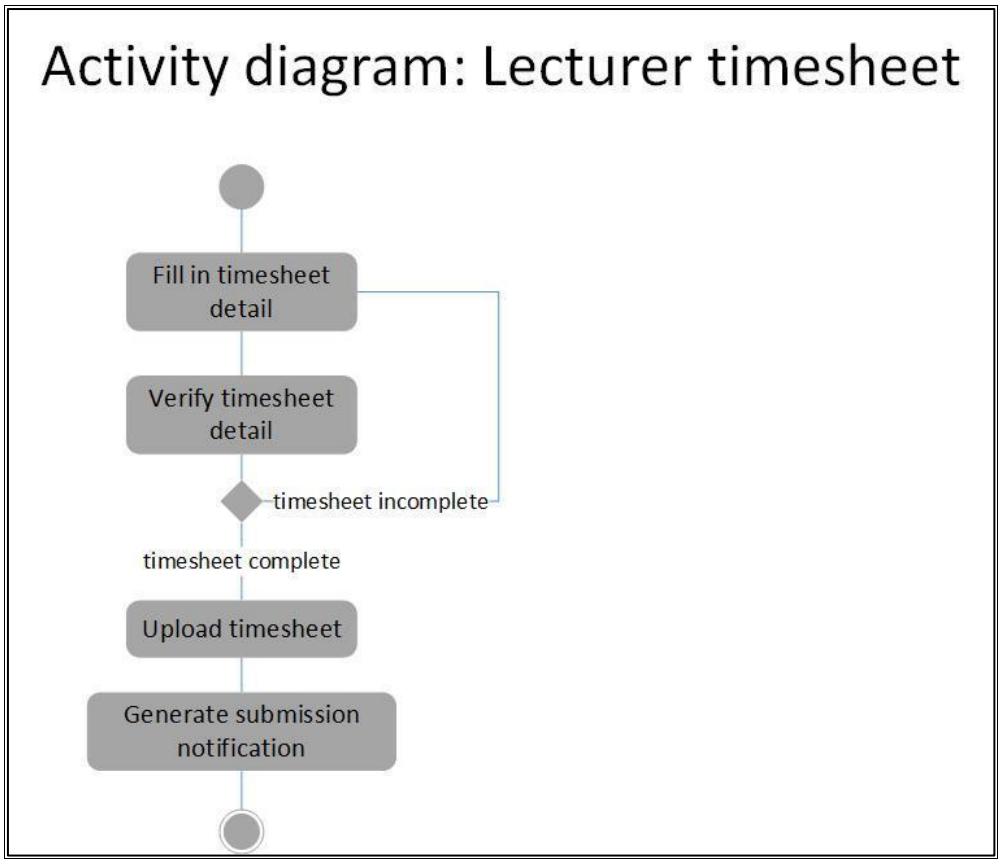

Figure 27 – Activity diagram: Lecturer timesheet

Summary

There are various formalised approaches that guide the entire software development life cycle, from requirements gathering to design, coding, testing, and maintenance. In addition to methods, teams apply specific techniques or practices to improve the quality and efficiency of their work. Software tools should be selected thoughtfully to align with the chosen development technique.

UML diagrams are used in software development to visualise, specify, construct, and document the structure and behaviour of software systems. They serve as a blueprint for understanding how a system works, making it easier to design, communicate, and manage complex projects.

The three UML diagrams that will be practically assessed in this module are:

- Use case diagram
- Class diagram (No attributes or operations required. Only classes, associations and multiplicity)
- Activity diagram

Software development methods, techniques, and tools help software development teams stay organised, reduce risks, maintain software quality, and deliver software that meets user expectations.

Self-Assessment Questions

1. Describe the importance of methods and techniques in software development.

2. Explain the importance of choosing the correct tools to conduct software development techniques.

3. Discuss the application of UML models in software development.

TOPIC 4
Software maintenance and documentation; quality management; and Artificial Intelligence (AI) and software development

4. INTRODUCTION

This topic relates to the following module outcome:

4. Demonstrate an understanding of the importance of software maintenance and documentation.
5. Identify modern software development and management platforms, tools, and services, and outline the nature of the support provided.
6. Demonstrate an understanding of the notion of quality in software and how a Quality Management System can provide the required organisational framework.

In this topic, you will gain knowledge in the following areas:

4.1 Software maintenance and documentation

4.2 Software quality management

4.3 Quality management framework

4.4 Artificial Intelligence (AI) and software development

Prescribed reading

Letaw, 2024:

- Non-functional Requirements (Page 35)

4.1 SOFTWARE MAINTENANCE AND DOCUMENTATION

4.1.1 Software maintenance

Software maintenance is a crucial phase in the software development life cycle (SDLC). It refers to the process of modifying and updating software applications after delivery to correct faults, improve performance, or adapt to a changed environment. Maintenance ensures that software continues to perform effectively and meet user needs long after its initial deployment (Dhaduk, 2023).

There are four primary types of software maintenance: corrective, adaptive, perfective, and preventive. Corrective maintenance deals with fixing bugs or errors. Adaptive maintenance involves updating software to work in new or changing environments. Perfective maintenance focuses on enhancing functionalities and user experience. Preventive maintenance aims at detecting and correcting hidden problems before they become serious (Mosia, 2024).

Maintaining software is vital to the success of any software application. As businesses grow and change, their software must evolve accordingly. Regular maintenance helps avoid system failures, security breaches, and performance issues, ensuring the software remains relevant, reliable, and efficient.

Despite its importance, software maintenance poses several challenges. These include understanding legacy code, dealing with outdated documentation, managing dependencies, and ensuring compatibility with new technologies. Maintaining old systems without disrupting business operations often requires significant time, resources, and expertise.

Software maintenance can account for up to 70% of the total cost of software over its lifecycle. The cost varies depending on the software's complexity, size, and the frequency of required updates. Preventive and perfective maintenance can be seen as long-term investments, reducing future costs by improving software stability and scalability (Ogheneovo, 2014).

According to Whitten and Bentley (2007:704), systems support comprises four ongoing activities, typically initiated by a problem or an opportunity related to the system:

- **Programme maintenance**: All systems contain errors or defects that may not have been detected during the testing phase. Programme maintenance involves correcting these errors or bugs that slipped through the systems development process. Such bugs may result from poorly validated

requirements, miscommunication, misinterpretation, or incorrect implementation.

- System recovery: In the event of system failure or data loss, system recovery involves restoring the system and its associated data to full functionality. This ensures minimal disruption and data integrity.
- Technical support: Even after training, users may require further assistance. This is often due to limited experience or unexpected scenarios not covered in training sessions. Technical support helps users navigate these challenges effectively.
- System enhancement: As business needs evolve and technology advances, systems must be updated. Enhancements are made to address new technical or business challenges or to incorporate modern technological developments.

The key objectives of system maintenance are centred around ensuring the continued effectiveness and reliability of software systems. One primary goal is to correct errors through planned and predictable modifications to existing systems, allowing faults to be addressed in a controlled and systematic manner. Another important objective is to preserve components that are functioning correctly, while preventing any ripple effects from changes that could negatively impact other parts of the system. In addition, maintenance aims to uphold system performance and prevent any degradation that might occur over time due to wear, data load, or evolving operational contexts. It is essential that all maintenance tasks are completed efficiently, without compromising the overall quality or reliability of the system.

4.1.2 Software documentation

The development team must assist the customer with the conversion process and provide end-user training. Another essential task is supplying the customer with system documentation (user manuals or other online sources) that will guide users in operating the new system. System documentation is often prepared by a systems analyst who specialises in writing user manuals and training guides. This documentation will also form part of the training sessions presented to the users.

Documentation plays an important role in effective software maintenance. It provides detailed information about the software's design, codebase, functionalities, and dependencies. Good documentation helps developers understand the system quickly, reducing the time and effort needed for troubleshooting and making changes.

There are several types of software documentation, including user manuals, system documentation, code documentation, and technical references. Each serves a different audience, users, system administrators, and developers, and is essential for training, troubleshooting, and maintaining consistency in future development (Atlassian, 2025).

Effective documentation should be clear, concise, up-to-date, and accessible. It should follow a consistent structure and style, include relevant diagrams or flowcharts, and provide examples where necessary. Version control and regular reviews are important to keep documentation aligned with the software's current state (Atlassian, 2025).

Many organisations make use of online help centres to provide support to end-users. These help centres are always accessible, offering users immediate assistance whenever needed. As the system or website is updated, the help content is also revised to reflect any changes. Typical features include "Getting Started" guides, Frequently Asked Questions (FAQs), troubleshooting steps, and community forums where users can engage with and assist each other.

Organisations also frequently include video tutorials to provide users with a more engaging and comprehensive learning experience, enhanced with multimedia. These tutorials help simplify complex tasks and improve user understanding. Notable examples of effective online support for end-users include Microsoft's Support Centre, Facebook's Help Centre, and Google Earth's Support Centre.

Modern tools and platforms can greatly assist in both software maintenance and documentation. Integrated development environments (IDEs), version control systems (e.g., Git), bug tracking tools (e.g., JIRA), and documentation generators (e.g., Doxygen, Sphinx) help streamline the maintenance process and keep documentation synchronised with the codebase.

Activity

Study the following link: https://support.apple.com/. Study the effectiveness of Apple's support centre.
Time allocation: 10 minutes

4.2 SOFTWARE QUALITY MANAGEMENT

Software Quality Management (SQM) promotes standardisation, proper documentation, and adherence to coding best practices, which in turn supports maintainability and scalability. Through quality audits, reviews, and continuous testing, SQM ensures that the final product is not only functional but also secure, efficient, and user-friendly. In an increasingly competitive software market, effective SQM is a key differentiator that can lead to long-term success and reduced operational risks.

Organisations choose to adopt Agile for a variety of reasons, one of the most significant being its impact on software quality. According to a study by HP involving 403 organizations that primarily use Agile methodologies, 52% of respondents agreed that Agile development increases the overall level of software quality within their organizations. This highlights the effectiveness in Agile methods in promoting better development practices and delivering higher-quality outcomes (Letaw, 2024:11).

SQM is a comprehensive process that ensures software meets the required standards and satisfies user needs. Conducting effective SQM involves a combination of planning, assurance, and control practices throughout the software development lifecycle (Draniceanu, 2024).

The following are examples of reliable software systems:

- A banking app processes thousands of transactions daily without crashing or losing data.
- A hospital's medical records system runs uninterrupted for months with minimal downtime.

The following are examples of usable software systems:

- A mobile app with intuitive navigation, accessible design, and easy-to-understand instructions.
- An e-learning platform that allows users of all skill levels to register, log in, and start a course without training.

The following is an example of software systems that are easy to maintain:

- Use of consistent coding standards and clear documentation that makes it easy to onboard new developers.

The process begins with quality planning, which lays the foundation for the entire quality management effort. In this phase, organisations define clear quality

objectives aligned with business goals and user requirements. Standards such as ISO 9001, IEEE, or CMMI are selected, and key quality metrics, like defect density or code coverage, are established. A comprehensive Quality Management Plan is then created, detailing roles, responsibilities, tools, procedures, and timelines to guide the quality efforts throughout the project lifecycle (Draniceanu, 2024; Naybour, 2022).

The next phase is Quality Assurance (QA), which focuses on preventing defects by improving development processes and ensuring teams adhere to defined standards. QA activities may include conducting process audits, organising peer reviews and walkthroughs of code and design, and training team members on best practices. These proactive measures help maintain consistency and prevent errors from entering the system (Garg, 2024).

Quality Control (QC), on the other hand, centres around identifying and correcting defects in the software product. This involves a wide range of testing activities, including unit testing, integration testing, system testing, and user acceptance testing. Automated testing tools are often employed to improve test coverage and efficiency. Defects are logged and tracked using bug-tracking systems, and regression testing ensures that recent changes have not adversely affected existing functionality (Garg, 2024).

Continuous monitoring and improvement are essential aspects of effective software quality management. Organisations regularly monitor quality metrics and key performance indicators (KPIs), collect user feedback after product releases, and conduct root cause analyses to address recurring issues. These insights are then used to refine development processes, improve tools, and enhance team capabilities, leading to a cycle of ongoing quality enhancement (BrowserStack, 2025).

Throughout the SQM process, various tools support quality efforts. Testing tools like Selenium and JUnit, CI/CD tools like Jenkins, and QA management platforms such as Jira and TestRail are commonly used to facilitate quality control and assurance activities. Code quality tools like SonarQube also help maintain high standards during development. Together, these practices and tools ensure a robust approach to managing software quality effectively (BrowserStack, 2025).

Activity

Study the following scenario:

A company recently launched a ride-sharing app. Users are complaining about frequent crashes, slow loading times, and poor customer support. There was no formal testing phase before release, and no documentation was provided to the support team.

Explain how quality assurance testing and complete documentation could have prevented these issues.

Time allocation: 15 minutes

4.3 QUALITY MANAGEMENT FRAMEWORK

In today's rapidly evolving, technology-driven landscape, software systems underpin virtually every sector of the economy. As the complexity and scale of software projects increase, ensuring high quality throughout the development process and in the final product has become essential. This is where Software Quality Management (SQM) frameworks prove invaluable.

A SQM Framework provides a structured and methodical approach to defining, evaluating, and enhancing software quality across the software development life cycle. It enables organisations to establish clear standards, processes, and best practices that support the consistent delivery of reliable, efficient, and user-centred software.

Implementing an SQM framework ensures that quality is not treated as an afterthought, but rather integrated into every phase of development, from requirements gathering and system design to coding, testing, and maintenance. These frameworks help teams identify and mitigate risks early, reduce defects, and enhance customer satisfaction.

Table 1 summarises the key Software Quality Management Frameworks used in many industries (Al-Daja, 2025; ASQ, 2025; Atlassian ITIL, 2025; Hayes, 2024; IEEE, 2025; ISO, 2025; Opperman, 2023; Rao, 2024; Son, 2024; and Sumrak, 2024):

Software Quality Management Frameworks

	Framework	Focus Areas	Primary Usage	Key Benefits
1	ISO/IEC 25010 (SQuaRE)	Product quality characteristics	Quality evaluation and benchmarking	Standardized quality metrics
2	ISO/IEC 12207	Software life cycle processes	Lifecycle process standardization	Comprehensive lifecycle coverage
3	CMMI	Process maturity & improvement	Process capability measurement	Structured process improvement
4	Six Sigma	Defect reduction, process improvement	Statistical quality control	Quantitative quality improvement
5	Total Quality Management (TQM)	Organization-wide quality improvement	Cultural and strategic quality practices	Employee involvement and quality culture
6	IEEE 730	Software Quality Assurance (SQA) plans	Standardizing SQA documentation	Consistency in QA planning
7	ITIL	IT service management quality	Service quality and delivery	Better IT service quality
8	ISO/IEC 15504 (SPICE)	Process assessment and improvement	Evaluating and improving processes	Objective process evaluation
9	Lean Software Development	Waste reduction, fast delivery	Improving flow and efficiency	Reduced development waste
10	Agile QA Frameworks	Continuous testing & collaboration	Agile software quality integration	Improved agility and quality
11	DevOps Quality Framework	Automation, CI/CD, feedback loops	DevOps pipeline quality assurance	Faster delivery with built-in quality
12	V-Model	Verification and validation	Testing-oriented development	Early defect detection

Table 1 – Software Quality Management Frameworks

Activity

Study the following paragraph:
In 1995, under the leadership of CEO Jack Welch, General Electric (GE) implemented Six Sigma as a core strategy to enhance operational efficiency and product quality.
Conduct more research on GE's adoption of Six Sigma.
Time allocation: 10 minutes

4.4 ARTIFICIAL INTELLIGENCE (AI) AND SOFTWARE DEVELOPMENT

Artificial Intelligence (AI) is increasingly transforming the landscape of software development. From writing and reviewing code to automating testing and managing projects, AI is playing an important role in how modern software is built and maintained. With continuous advancements in machine learning and natural language processing, developers now have powerful tools at their fingertips that improve productivity and accuracy, allowing them to focus on more strategic and creative aspects of their work.

One of the most visible impacts of AI in software development is in code generation and completion. Tools like GitHub Copilot and ChatGPT can analyse code patterns and offer suggestions or even write full blocks of code based on natural language inputs. This significantly reduces the time spent on repetitive tasks and helps developers work more efficiently (Odazie, 2025).

AI can also support the testing phase by automatically generating test cases, identifying bugs, and recommending fixes. These tools learn from vast datasets and previous errors, which allows them to predict potential issues before they become problems (Sujatha, 2025).

Debugging, a traditionally time-consuming process, is also becoming more streamlined thanks to AI. Intelligent systems can scan through codebases in real-time, detect errors, and provide insights into their possible causes. This not only speeds up the debugging process but also helps in maintaining higher code quality. AI contributes to better project management by analysing developer productivity, predicting delivery timelines, and identifying roadblocks. Natural language processing enables the summarisation of documentation and user

requirements, making collaboration and communication smoother within teams (Microsoft, 2024).

An important benefit of AI in software development is the rise of natural language interfaces. These allow both developers and non-developers to interact with software systems using everyday language. This democratises development by opening the door for more people to contribute to software creation, even without formal programming skills. As a result, organisations can develop custom solutions and automate processes with greater ease (Tewari, 2023).

The integration of AI into software development brings several key benefits. It increases efficiency, reduces the likelihood of bugs, lowers costs, and enables more innovation. However, it is not without challenges. AI models can carry biases from the data they're trained on, potentially leading to flawed outputs. Over-reliance on AI could also cause developers to lose touch with core coding skills. There are concerns about security and intellectual property when using AI-generated code (WSU, 2025).

Activity

Study the following paragraph:
Forecast is an AI-powered project management tool that provides a unified platform to simplify project creation, budgeting, resource allocation, task management, invoicing, and reporting through automation and smart insights.
Conduct more research on *Forecast*.
Time allocation: 10 minutes

Summary

Software maintenance and documentation are ongoing responsibilities that ensure the long-term effectiveness and usability of software systems. While often overlooked in favour of new development, they are essential for sustaining functionality, minimising downtime, and meeting evolving user requirements. A well-maintained and well-documented software product is easier to support, upgrade, and trust.

Software Quality Management (SQM) is essential for ensuring that software products meet both customer expectations and industry standards. It involves a systematic process of planning, assurance, and control activities that guide software development towards achieving high quality. By implementing quality management practices, organisations can detect and address defects early in the development cycle, reducing the cost and time associated with fixing issues after deployment. This proactive approach not only enhances product reliability and performance but also contributes to customer satisfaction and trust in the software.

SQM frameworks foster collaboration, accountability, and continuous improvement across development teams. SQM frameworks support compliance with industry standards, lower development costs, and strengthen an organisation's competitive position in the market. Adopting a SQM Framework is not merely about meeting technical benchmarks, it is about building trust, delivering long-term value, and ensuring sustainable success in software-driven initiatives.

Artificial Intelligence (AI) is not replacing software developers; it is empowering them. By handling repetitive tasks and offering intelligent assistance, AI enables developers to focus on innovation and problem-solving. The future of software development will be shaped by this collaboration between human ingenuity and artificial intelligence, leading to smarter, more efficient, and more inclusive technology.

Self-Assessment Questions

1. Describe the importance of information system quality in software development.

2. Explain system recovery as part of software maintenance.

3. Discuss the importance of complete software documentation.

4. Discuss the application of a quality framework in software development.

5. Discuss the latest trends in Artificial Intelligence (AI) adoption for software development.

References

Agile Alliance. 2025. The Agile Manifesto. Online available: https://www.agilealliance.org/agile101/the-agile-manifesto/ [Accessed: 12 March 2025].

Al-Daja, D. 2025. Lean Software Development: Maximizing Efficiency and Delivering Value. Online available: https://www.linkedin.com/pulse/lean-software-development-maximizing-efficiency-value-dania-al-daja-pik0e [Accessed: 07 April 2025].

Allbee, B. 2018. Hands-on Software Engineering with Python. Packt Publishing. Online available: https://www.scholartext.com/reader/docid/88864711/page/1?searchterm=Handson%20Software%20Engineering%20with%20Python [Accessed: 07 February 2025].

ASQ. 2025. Total Quality Management. Online available: https://asq.org/quality-resources/total-quality-management [Accessed: 07 April 2025].

Atlassian. 2025. How to create effective product documentation. Online available: https://www.atlassian.com/work-management/knowledge-sharing/documentation/product-documentation [Accessed: 07 April 2025].

Atlassian. 2025. Waterfall Methodology: A Comprehensive Guide. Online available: https://www.atlassian.com/agile/project-management/waterfall-methodology#:~:text=The%20Waterfall%20methodology%20works%20best%20for%20project,with%20last%2Dminute%20requirements%20is%20suitable%20for%20Waterfall. [Accessed: 12 March 2025].

Atlassian ITIL. 2025. A guide to ITIL and its place in modern ITSM. Online available: https://www.atlassian.com/itsm/itil [Accessed: 07 April 2025].

Awati, R. & Gillis, A.S. 2024. What is systems development life cycle? Online available: https://www.techtarget.com/searchsoftwarequality/definition/systems-development-life-cycle [Accessed: 12 March 2025].

Axure. 2025. Online available: https://www.axure.com/ [Accessed: 12 March 2025].

Bigelow, S.J. 2025. The history of cloud computing explained. Online available: https://www.techtarget.com/whatis/feature/The-history-of-cloud-computing-explained [Accessed: 07 February 2025].

BrowserStack. 2025. How to ensure an efficient Software Quality Management Process. Online available: https://www.browserstack.com/guide/efficient-software-quality-management-process [Accessed: 07 April 2025].

Cardozo, S. 2024. Our 20 Recommended Software Development Tools for 2024. Online available: https://loopstudio.dev/best-software-development-tools/#:~:text=JIRA%20lets%20you:%20issue%20tracking%2C%20project%20management%2C,a%20comprehensive%20solution%20for%20software%20development%20projects. [Accessed: 12 March 2025].

Dhaduk, H. 2023. What is Software Maintenance: Importance, Types, Phases, and Models. Online available: https://www.simform.com/blog/software-maintenance/ [Accessed: 07 April 2025].

Draniceanu, A. 2024. What is Software Quality Management? Online available: https://thectoclub.com/quality-engineering-planning-strategy/what-is-software-quality-management/ [Accessed: 07 April 2025].

Garg, R. 2024. QA vs QC: Understanding the Key Differences and Their Roles in Software Development. Online available: https://www.frugaltesting.com/blog/qa-vs-qc-understanding-the-key-differences-and-their-roles-in-software-development [Accessed: 07 April 2025].

Gupta, R. 2024. 10 Principles of Software Development You Must Know! Online available: https://www.turing.com/blog/principles-of-software-development-guide [Accessed: 07 February 2025].

Hayes, A. 2024. What Is Six Sigma? Concept, Steps, Examples, and Certification. Online available: https://www.investopedia.com/terms/s/six-sigma.asp [Accessed: 07 April 2025].

HP Tech Takes. 2018. Computer History: A Timeline of Computer Programming Languages. Online available: https://www.hp.com/us-en/shop/tech-takes/computer-history-programming-languages [Accessed: 07 February 2025].

IBM. 2024. What is the software development life cycle (SDLC)? Online available: https://www.ibm.com/think/topics/sdlc [Accessed: 12 March 2025].

Icasas, P. 2024. Spiral model: The Spiral Project Management Method. Online available: https://birdviewpsa.com/blog/upward-spiral-spiral-project-management-method/ [Accessed: 12 March 2025].

IEEE. 2025. IEEE Standard for Software Quality Assurance Processes. Online available: https://standards.ieee.org/ieee/730/5284/ [Accessed: 07 April 2025].

ISO. 2025. Online available: https://www.iso.org/search.html?PROD_isoorg_en%5Bquery%5D=software%20quality [Accessed: 07 April 2025].

Stryker, C. & Kavlakoglu, E. 2024. What is artificial intelligence (AI)? Online available: https://www.ibm.com/think/topics/artificial-intelligence [Accessed: 07 February 2025].

Jewel, U. 2023. 5 Software Development Principles That Should Be Embraced Daily. Online available: https://thebulbafrica.medium.com/5-software-development-principles-that-should-be-embraced-daily-75a6f6cc00ff [Accessed: 07 February 2025].

Khan, S.M.A. 2023. A Comprehensive Journey through the History of Software Engineering. Online available: https://sardarmudassaralikhan.medium.com/a-comprehensive-journey-through-the-history-of-software-engineering-66c590126a97 [Accessed: 07 February 2025].

Lee, R.Y. 2013. Software Engineering: a hands-on approach. Atlantis Press. Online available: https://www.amazon.com/Software-Engineering-Hands-Roger-Lee/dp/9462390053 [Accessed: 07 February 2025].

Letaw, L. 2024. Handbook of Software Engineering Methods. 2nd Edition. Online available: https://open.oregonstate.education/setextbook/ [Accessed: 26 March 2025].

Levrel, M. 2025. 17 Best AI Code Generators for 2025. Online available: https://www.qodo.ai/blog/best-ai-code-generators/ [Accessed: 12 March 2025].

Lomio, M. K. & Mejidana, M. 2023. Beginner's Guide to Rapid Application Development (RAD). Online available: https://medium.com/@myelmarc/sdlc-

methodology-101-a-beginners-guide-to-rapid-application-development-rad-492ab88de7ef [Accessed: 12 March 2025].

Michalowski, M. 2025. Top 27 Software Development Tools & Platforms [2025 List]. Online available: https://spacelift.io/blog/software-development-tools [Accessed: 12 March 2025].

Microsoft. 2024. Debug your code with AI. Online available: https://www.microsoft.com/en-us/microsoft-copilot/for-individuals/do-more-with-ai/general-ai/debug-code [Accessed: 07 April 2025].

Microsoft. 2025. Microsoft Azure. Online available: https://azure.microsoft.com/en-us [Accessed: 07 February 2025].

Mosia, B. 2024. Understanding the 4 Main Categories of Software Maintenance. Online available: https://www.scrums.com/guides/the-4-main-categories-of-software-maintenance [Accessed: 07 April 2025].

National Archives. 2025. Online available: https://education.blogs.archives.gov/2015/10/08/upcoming-events-for-educators-at-the-national-archives-at-new-york-city/1260_original/ [Accessed: 07 February 2025].

Naybour, P. 2022. Describe the four main components of a quality management process. Online available: https://www.parallelprojecttraining.com/blog/describe-the-four-main-components-of-a-quality-management-process/ [Accessed: 07 April 2025].

Odazie, D. 2025. GitHub Copilot vs. ChatGPT: Developer AI Tools Comparison. Online available: https://spacelift.io/blog/github-copilot-vs-chatgpt [Accessed: 07 April 2025].

Ogheneovo, E.E. 2014. On the Relationship between Software Complexity and Maintenance Costs. *Journal of Computer and Communications*, vol. 2, no. 14. Online available: https://www.scirp.org/journal/paperinformation?paperid=51631 [Accessed: 07 April 2025].

OMG. 2025. bout the Unified Modeling Language Specification Version 2.5.1 Online available: https://www.omg.org/spec/UML [Accessed: 26 March 2025].

Okeke, F. 2022. The 12 Best IDEs for Programming. Online available: https://www.techrepublic.com/article/best-ide-software/ [Accessed: 07 February 2025].
Opperman, A. 2023. What Is the V-Model in Software Development? Online available: https://builtin.com/software-engineering-perspectives/v-model [Accessed: 07 April 2025].

Raeburn, A. 2025. Extreme programming (XP) gets results, but is it right for you? Online available: https://asana.com/resources/extreme-programming-xp [Accessed: 12 March 2025].

Rao, R. 2024. CMMI: The Ultimate Guide to Transforming Your Organization's Processes. Online available: https://buzzclan.com/digital-transformation/cmmi/ [Accessed: 07 April 2025].

Sharma, S. 2024. 5 Best Programming Languages for App Development in 2025. Online available: https://www.linkedin.com/pulse/5-best-programming-languages-app-development-2025-soma-sharma-zi9rc [Accessed: 07 February 2025].

Smartdraw. 2025. Online available: https://www.smartdraw.com/software-engineering/ [Accessed: 12 March 2025].

Sommerville, I. 2011. Software Engineering 9th edition. Online available: https://uoitc.edu.iq/images/documents/informatics-institute/exam_materials/Software%20Engineering%20(9th%20Edition)%20by%20Ian%20Sommerville.pdf [Accessed: 07 February 2025].

Son, H. 2024. Agile QA Process: Principles, Steps, and Best Practices. Online available: https://www.testrail.com/blog/agile-qa-best-practices/ [Accessed: 07 April 2025].

Sujatha, R. 2025. Online available: https://www.digitalocean.com/resources/articles/ai-testing-tools [Accessed: 07 April 2025].

Sumrak, J. 2024. DevOps vs. CI/CD: Complete Guide to Better Software Delivery. Online available: https://launchdarkly.com/blog/devops-vs-cicd/ [Accessed: 07 April 2025].

Swaine, M.R. & Freiberger, P.A. 2025. ENIAC. Online available: https://www.britannica.com/technology/ENIAC [Accessed: 07 February 2025].

Tewari, G. 2023. Embracing AI And Natural Language Interfaces. Online available: https://www.forbes.com/councils/forbesbusinesscouncil/2023/07/11/embracing-ai-and-natural-language-interfaces/ [Accessed: 07 April 2025].

Valacich, J.S.; George, J.F. & Hoffer, J. 2015. Essentials of systems analysis and design, global edition. 6th edition. Pearson Education.

Victor, M.F. 2025. The Spiral Model. Online available: https://ultimatesdlc.com/spiral-model/ [Accessed: 12 March 2025].

Whitten, J.L. & Bentley, L.D. 2007. Systems analysis and design methods. 7th edition. Boston: McGraw-Hill Higher Education.

WSU. 2025. Challenges of AI. Online available: https://provost.wsu.edu/challenges-of-ai/ [Accessed: 07 April 2025].

Answers to Self-Assessment Questions

TOPIC 1 – SELF-ASSESSMENT ANSWERS

1. Describe the emergence of the field of software engineering.

The field of software engineering emerged in response to the growing complexity and scale of software systems. It was developed to apply engineering principles to software development, ensuring that software is reliable, maintainable, and efficient. The field gained momentum in the 1960s when organisations faced challenges with large, error-prone programs.

2. Discuss why older programming languages are still relevant today.

Older programming languages like COBOL and Fortran remain relevant today because they support legacy systems that are still operational in industries like banking and government. These languages are trusted for their stability, have proven reliability, and their maintenance ensures continued functionality of critical systems.

3. Explain why software developers need sound methodologies.

Software developers need sound methodologies to structure the development process, ensure quality, and manage complexity. Methodologies like Agile and Waterfall provide guidance on planning, coding, testing, and delivery, which improves productivity and minimises the risk of project failure.

4. Describe the differences between machine learning and deep learning.

Machine learning involves algorithms that improve with experience by identifying patterns in data, whereas deep learning is a subset of machine learning that uses artificial neural networks with multiple layers to handle complex data relationships. Deep learning models are especially suited for tasks like image and speech recognition.

5. Discuss the importance of 'reuse' as a software development principle.

'Reuse' in software development promotes using existing code and components, which saves time, reduces errors, and ensures consistency. Reusing tested code speeds up development and enhances the reliability of new software products, reducing costs and resource usage.

6. Discuss the impact of cloud computing on software development.

Cloud computing has transformed software development by offering scalable, on-demand infrastructure and services. Developers can deploy applications faster, collaborate globally, and reduce costs by leveraging cloud resources, which also enable continuous integration and delivery of updates.

TOPIC 2 – SELF-ASSESSMENT ANSWERS

1. Describe the sequential nature of the Waterfall model for systems development.

The Waterfall model follows a sequential approach where each phase; requirements, design, implementation, testing, deployment, and maintenance; must be completed before the next begins. This linear process makes it easier to track progress but offers limited flexibility for changes during development.

2. Discuss why agile methodologies are gaining attention in the field of software development.

Agile methodologies are gaining attention because they focus on flexibility, collaboration, and delivering value quickly. Unlike traditional methods, Agile allows for regular feedback, incremental delivery, and adaptation to changing requirements, which is crucial in today's fast-paced software environment.

3. Explain the importance of the daily SCRUM meeting used by development teams.

The daily SCRUM meeting is vital for keeping development teams aligned and informed. It promotes communication, identifies obstacles early, and ensures accountability. These short, focused meetings help teams adapt to changes and keep projects on track.

4. Discuss some of the best software tools on the market for project management.

Some of the best project management software tools on the market include Jira, Trello, and Asana. These tools support task tracking, collaboration, and progress monitoring, helping teams to manage projects efficiently, meet deadlines, and adapt to changing priorities.

TOPIC 3 – SELF-ASSESSMENT ANSWERS

1. Describe the importance of methods and techniques in software development.

Methods and techniques are crucial in software development because they provide structured approaches to planning, designing, testing, and maintaining software. They ensure that software meets user needs, reduces errors, and delivers reliable and maintainable products.

2. Explain the importance of choosing the correct tools to conduct software development techniques.

Choosing the correct tools for software development techniques is important because the right tools enhance productivity, accuracy, and collaboration. Suitable tools support effective implementation of techniques, reducing development time and errors while improving code quality and maintainability.

3. Discuss the application of UML models in software development.

UML (Unified Modelling Language) models help software developers visualise and document system designs. They provide a standard way to represent system architecture, processes, and data flows, which improves communication among stakeholders and guides development efforts efficiently.

TOPIC 4 – SELF-ASSESSMENT ANSWERS

1. Describe the importance of information system quality in software development.

Information system quality is crucial in software development as it ensures systems are reliable, secure, and meet user requirements. High-quality systems reduce errors, improve user satisfaction, and support business operations effectively.

2. Explain system recovery as part of software maintenance.

System recovery as part of software maintenance involves restoring systems to normal operation after failures, such as hardware malfunctions or data corruption. It is vital for minimising downtime, protecting data, and ensuring business continuity.

3. Discuss the importance of complete software documentation.

Complete software documentation is important because it supports understanding, maintenance, and future development. It provides a clear reference for developers, testers, and users, ensuring consistency and reducing errors.

4. Discuss the application of a quality framework in software development.

Applying a quality framework, such as ISO 9001 or CMMI, in software development helps standardise processes, improve efficiency, and ensure products meet high-quality standards. These frameworks guide teams in delivering reliable and maintainable software.

5. Discuss the latest trends in Artificial Intelligence (AI) adoption for software development.

The latest trends in AI adoption for software development include using AI-powered coding assistants, automated testing, and intelligent code reviews. AI

enhances productivity, accelerates development, and supports better decision-making through predictive analytics.