**Power Learn Project**

**Software Engineering assignment**

**Siyabonga T. Pika**

**Software engineering vs traditional programming**

Software engineering embraces a methodological, structured, and measurable approach to the creation, management, and maintenance of software. It requires the application of specific software development concepts to produce reliable, effective, and high-quality software. It ensures that software is built in a planned and organized way by adhering to a defined process. Following software engineering’s best practices, standards, and guidelines promotes quality and uniformity across the software development lifecycle. Software development projects are evaluated based on quantifiable criteria that measure their quality, performance, and advancement.

Traditional programming typically focuses on creating code to do specific tasks or address specific problems, whereas software engineering focuses on the full software development lifecycle, including requirements gathering, design, development, testing, deployment, and maintenance. Software engineering makes use of formal frameworks and processes to direct the development process, promising that every facet of software development is considered and efficiently managed. On the other hand, in traditional programming programmers may create code based on urgent requirements without a structured strategy or procedure and not adhering to formal approaches.

**The Software Development Life Cycle**

The Software Development Life Cycle (SDLC) is an organized method for creating software applications. It consists of multiple phases, each with its own set of tasks and outputs. Planning is the first stage of the project, during which feasibility studies and goal identification are used to determine the project's scope and purpose. The project plan that results from this step specifies the required materials, budget, and timetables. The requirements analysis step, which comes after planning, entails obtaining and examining business requirements to understand the intended functionalities of the system. Deliverables such as a requirements specification document and use cases are the products of stakeholders working together to capture requirements and generate a requirements specification.

The Design phase comes next. It focuses on developing the system architecture and components. In this phase, design documents, system architecture diagrams, data flow diagrams, entity-relationship diagrams, and user interface designs are the results of developing architectural diagrams, database schemas, data models, and user interface planning. The next stage is called Implementation (or Coding), during which the design is turned into usable software. Writing code, completing unit tests, and integrating various modules are all part of this process, which produces deliverables including source code, generated code, and unit test cases.

To make sure the program satisfies requirements and is error-free, the testing step seeks to locate and correct flaws. Testing is done in several ways, such as acceptance, system, integration, and unit testing. Test plans, test cases, defect reports, and test summary reports all include documentation of the outcomes. Following testing, the software is released into the production environment during the deployment phase so that end users can utilize it. Deliverables for this phase include a deployment strategy, training materials, user manuals, and the delivered software. It also entails setting up the deployment environment, deploying the software, training users, and moving to operational status.

Lastly, after deployment, the maintenance step makes sure the software keeps working correctly and effectively. The activities involve monitoring the software for any problems, delivering updates and improvements, resolving bugs, and carrying out regular maintenance duties. The results include maintenance reports, patches, and updates. Depending on the SDLC model - Waterfall, Agile, or Spiral, for example - these phases can be carried out in a sequential or iterative approach. Each phase builds on the results of previous ones to ensure a methodical approach to software development.

**Agile vs Waterfall model**

The Agile and Waterfall models are the primary approaches to software development, each with unique benefits, disadvantages, and appropriate uses. The Waterfall model adopts a sequential approach of the requirements, design, implementation, testing, deployment, and maintenance phases as discussed in the SDLC discussion above. It is a systematic and predictable practice since each step must be finished before going on to the next. This paradigm functions best for projects with well-defined needs and boundaries, including those in the public sector or other regulated fields where strict documentation is necessary and changes are kept to a minimum.

The Agile model, on the other hand, is a flexible and iterative methodology that prioritizes customer feedback, teamwork, and quick, short software releases. Agile divides projects into manageable chunks called sprints, which usually last two to four weeks and allow for frequent plan revision and re-evaluation. Teams can react swiftly to changes and raise the quality of their products because of this model's support for continuous integration and testing. Agile is best suited for projects that require regular revisions or have changing requirements, as those in software product development, tech startups, or high-uncertainty situations.

The two models differ mostly in how flexible and adaptable they are. Waterfall lacks tools for accepting feedback until late in the process, which can cause problems if initial requirements are misinterpreted or if adjustments are needed mid-project. Agile's iterative cycles enable ongoing customer interaction and iterative improvement, which can improve final products and increase customer satisfaction. However, in larger businesses where a great degree of documentation is required or where processes are deeply embedded, implementing Agile can be difficult.

**Requirements engineering**

Requirements engineering is an important step in the software development lifecycle. It entails determining, capturing, and upholding stakeholder demands and specifications to promote the effective creation of software systems. It acts as a cornerstone for all other project stages, directing design, development, and testing activities to be in line with user requirements and organizational objectives. The steps usually involved in the process, include requirements gathering, analysis, definition, validation, and management.

Requirements engineers collect data from users, clients, and other interested parties during requirements gathering to understand their requirements and limitations. Surveys, workshops, interviews, and an analysis of the current documentation and systems may all be part of this phase. After gathering, the requirements are examined to identify priorities, settle disagreements, and ensure viability. This analysis assists in determining any technological or resource constraints as well as the distinction between essential and non-essential requirements.

After analysis, the requirements are formally recorded in a specification that stakeholders and developers can refer to at any point during the project. This document usually contains non-functional needs that describe the system's operational features, such as performance, security, and usability, in addition to functional requirements that specify what the system should be able to do. Next, validation makes sure that the criteria are consistent, comprehensive, and appropriately represent the demands of the stakeholders. Prototypes, evaluations, and modelling techniques may be used to verify that the requirements are accurate and realistic.

As a project progresses, requirements are tracked and updated as part of the continuous requirements management process. The requirements may need to be adjusted in response to shifts in the company's goals, the state of the market, or technological limitations. Maintaining alignment between the project's objectives and its execution is made possible by effective requirements management, which ensures that these modifications are methodically assessed and incorporated.

Requirements engineering lowers the possibility of expensive redesigns and project failures by clearing up misconceptions and misplaced expectations. It helps with better planning, more accurate cost estimation, and enhanced communication between team members and stakeholders by offering a defined and agreed upon foundation for the project. These benefits eventually translate into higher quality software and improved customer satisfaction.

**Software design principles (Modularity)**

Modularity can be viewed as a fundamental design concept that supports the scalability and maintainability of software systems. It refers to the division of a software system into discrete, independent modules, each in charge of a certain area of the system's operation. These modules can be built, tested, and maintained independently of one another because of their clearly specified interfaces. This method improves the codebase's structure and organization, making it easier to maintain and understand. Developers can work on certain components without impacting the system by separating various functionalities into modules. This makes it easier to find and repair errors, add new features, or update current ones.

Since it reduces complexity and encourages reusability, modularity greatly enhances software systems' maintainability. Each module is simpler to comprehend and alter if it is made to carry out a single task or a small number of closely related tasks. Because of this division of responsibilities, modifications made to one module usually have little influence on the system, reducing the possibility of unforeseen consequences. Furthermore, because individual modules may be verified separately before being integrated into a larger system, modular programming is simpler to test.

Modularity also improves scalability since it enables systems to develop and grow in a controlled way. The addition of new modules or the upgrading of current ones can be done with little effect on other system components. This modular design facilitates dispersed development, which speeds up development and allows for simultaneous advancement by allowing multiple teams to work on distinct modules at the same time. Modular systems also make it easier to adapt to evolving needs or technological advancements because individual modules can be upgraded or replaced without requiring a total system redesign.

**Testing in software engineering**

In software development, testing is essential since it ensures that software is secure, reliable, and of the highest quality. Early defect and issue detection lowers the cost and effort of fixing problems later in the development process. By confirming that the program satisfies user needs and functions properly in a variety of contexts, effective testing enhances the overall user experience. Comprehensive testing also improves the software's performance, security, and flexibility, ensuring long-term success and user satisfaction.

Unit testing is the initial stage that concentrates on specific software modules or components. Every unit - usually the tiniest segment of code, such a function or method - is examined separately to ensure that it functions as intended. Unit testing makes the debugging process more effective by assisting in the early detection and correction of errors in the development cycle.

After unit testing, integration testing combines separate components and tests them collectively. The purpose of this testing level is to find problems that result from the interplay between integrated units. It ensures that data moves across the modules accurately and that they function together as intended. Issues with data formats, communication protocols, and module interfaces that may go undetected in unit testing can be found via integration testing.  
The next step is system testing, which involves testing the integrated software system. At this stage, it is confirmed that the system complies with the requirements. It entails putting the software through its paces in a setting that closely resembles the production setting. System testing ensures that the program performs as expected in real-world scenarios by examining a variety of factors, including functionality, performance, security, and usability.

The last stage, acceptance testing, includes assessing the program from the viewpoint of the end users. This testing, which is usually carried out by the client or end user, confirms that the program satisfies their needs and specifications. The program is tested in controlled environments and real-world circumstances during the alpha and beta testing phases of acceptance testing, respectively. When acceptance testing is completed successfully, the software is prepared for use.

**Version control systems**

Software developers need version control systems to track and manage code changes over time. Version control systems preserve a history of any modifications made to the software, enabling several developers to work together on a project without erasing each other's contributions. This history makes it possible for engineers to understand the software's evolution, compare different versions to discover specific changes, and roll back to earlier versions if a new modification produces errors. Version control systems also support branching and merging that enable teams to work on several features or fixes at once and then easily incorporate them into the main project.

Version control systems are important in software development because they make it easier for software developers to collaborate, improve project management, and promote code integrity. They offer a safety net that lowers the risk of data loss and assists in auditing and accountability by maintaining a thorough record of modifications. Additionally, they simplify operations, facilitating better teamwork management and coordination regardless of developers’ geographical location.

Popular version control systems are Mercurial, Subversion, and Git. Git is the most popular version control system that is recognized for its distributed architecture that enables every developer to have an identical copy of the repository. Performance is enhanced by this approach, which also offers strong branching and merging support. Additionally, Git works well with systems like GitHub and GitLab, which provide extra functionality such as issue tracking, pull requests, and means for continuous integration and continuous deployment (CI/CD).  
Subversion is a centralized version control system that enables one repository to be kept up to date on a server. It may not offer the same degree of flexibility and offline capabilities as Git, despite having a simpler paradigm and being easier to set up for smaller teams. On the other hand, Mercurial is a distributed version control system that is popular for its performance and usability, which might be a decent substitute for Git in some applications. It is designed to manage extremely large projects effectively. Its decentralized architecture makes it possible for several developers to work on a project at once without obstructing one another's advancement. It is a high-performance tool that works well with both small and big codebases. Furthermore, Mercurial promotes strong data integrity and facilitates broad collaboration with features such as change tracking, history visualization, and tool integration.

**Software project management (project manager)**

A software project manager is essential to the effective planning, carrying out, and completion of software projects. Project managers’ main duty is to supervise the planning, carrying out, and finishing of software development projects, making certain that the projects are completed within the allotted budget, on schedule, and with the required level of quality. This entails a broad range of activities, beginning with working with stakeholders to define the project's objectives, deliverables, and scope. They oversee drafting thorough project plans that specify the schedule, distribution of resources, and interdependencies between tasks. They must collaborate with cross-functional teams, such as developers, designers, and testers, to make sure that everyone is on the same page and working toward the same objective. For this reason, effective communication is essential.

Risk management is one of the main duties of a software project manager. This means that early in the project, possible risks must be identified, and mitigation methods must be created to meet them. To keep the project on schedule, they must also constantly assess the project's progress and adjust as needed. This involves overseeing any alterations to the project's scope, which frequently necessitates striking a balance between the conflicting demands of stakeholders and making sure that the project's budget or schedule are not jeopardized.

Project managers deal with several difficulties. Managing stakeholder expectations is one of the biggest challenges, and it can be especially challenging when there are competing priorities or requirements for the project change. They also must cope with the technical difficulties of developing software, which frequently entail integrating new technologies and resolving unanticipated problems. Another issue is maintaining the project team's motivation and output, particularly in the face of pressure and close deadlines. Furthermore, it can be challenging to maintain cohesiveness and clarity in a geographically scattered team through effective communication, necessitating the usage of a variety of collaboration technologies and strategies.

**Software maintenance**

After software is first deployed, it can still need to be updated to fix bugs, enhance functionality, or adjust to new settings. This process is known as software maintenance. This phase is important because it ensures that the software will continue to be efficient, reliable, and functional over time to adapt to changing user requirements and technology breakthroughs. Remedial, adaptive, perfective, and preventive maintenance are the four main categories of maintenance activities.

Remedial maintenance entails finding and repairing software flaws or errors. These problems can range from little bugs to significant faults that impair functioning and can be found through testing, monitoring, or user feedback. Updating software to keep it compatible with new or changing environments - such as operating system updates, hardware upgrades, or software integration - is known as adaptive maintenance. The goal of perfective maintenance is to make the program better by either adding new features or optimizing its performance in response to user requests or evolving specifications. This kind of maintenance keeps the program current and functional by ensuring that it always meets user expectations. Software modifications are part of preventive maintenance aiming to counteract issues before they arise. This can involve reorganizing the software to make it easier to maintain and less prone to problems, updating the documentation, or optimizing the code.

Since maintenance ensures the long-term survival and efficacy of software applications, it is a central component of the software lifecycle. Software can become outdated, inefficient, and more error-prone without routine maintenance, which can raise expenses and irritate users. It contributes to upholding the value of the software investment by ensuring that the software meets user needs and corporate objectives throughout time. Additionally, it permits software to be improved gradually, which can be less disruptive and more economical than total overhauls or replacements. Therefore, for software to be managed successfully and last a long time, a strong maintenance strategy must be implemented.

**Ethical considerations in software engineering**

Software engineers contemplate ethical considerations because the software they produce may have extreme effects on people, companies, and society. Data security and privacy are key concerns with an emphasis on defending the integrity and confidentiality of user data against breaches and illegal access. Another area is intellectual property, where it's important to uphold licensing agreements, respect rights, and refrain from plagiarism. Considering the need for engineers to be conscious of and take steps to reduce biases that can produce unjust results, algorithmic bias also poses ethical problems. Honesty regarding the capabilities and constraints of software is necessary for transparency and accountability, preventing users or stakeholders from being misled. In important industries such as healthcare, banking, and transportation, where software errors can have severe consequences, ensuring security and reliability becomes particularly critical.

**References**

Mishra, A. & Alzoubi, Y.I. (2023) Structured software development versus agile software development: a comparative analysis. *Int J Syst Assur Eng Manag* **14**, 1504–1522. <https://doi.org/10.1007/s13198-023-01958-5>

Web resources

[Agile vs. Waterfall | Pros, Cons, and Key Differences (productplan.com)](https://www.productplan.com/learn/agile-vs-waterfall/)

[Ethical Considerations in Software Engineering: Balancing Innovation and Responsibility | by Mikel Vu | A better leader | Medium](https://medium.com/a-better-leader/ethical-considerations-in-software-engineering-balancing-innovation-and-responsibility-995d610e45e4)

[Introduction to Modularity in System Design (prepbytes.com)](https://www.prepbytes.com/blog/system-design/introduction-to-modularity-in-system-design/)

[Modularity and its Properties - GeeksforGeeks](https://www.geeksforgeeks.org/modularity-and-its-properties/)

[Requirements Engineering Process in Software Engineering - GeeksforGeeks](https://www.geeksforgeeks.org/software-engineering-requirements-engineering-process/)

[Role and Responsibilities of a software Project Manager - Software Engineering - GeeksforGeeks](https://www.geeksforgeeks.org/software-engineering-role-and-responsibilities-of-a-software-project-manager/)

[Software Development Life Cycle (SDLC) - GeeksforGeeks](https://www.geeksforgeeks.org/software-development-life-cycle-sdlc/#full-form-of-sdlc)

[Software Development Process: Methodologies And Tools [With Best Practices] (lambdatest.com)](https://www.lambdatest.com/learning-hub/software-development-process)

[Types of Software Testing - GeeksforGeeks](https://www.geeksforgeeks.org/types-software-testing/)

[What is version control | Atlassian Git Tutorial](https://www.atlassian.com/git/tutorials/what-is-version-control)