<u>**ASSIGNMENT 2**</u>

Software Engineering is a branch of computer science that deals with the design, development, testing, and maintenance of applications using engineering principles and programming language knowledge.

How does it differ from traditional programming?

Traditional Programming refers to conventional methods and paradigms of writing software codes characterized by writing explicit instructions for the computer to perform, focusing on how task are accomplished.

	Difference categorization	**Software Engineering**	**Traditional programming**
	Scope and Focus	Focuses on producing high-quality software that meets user needs, is maintainable, scalable, and reliable	Concerned with the act of writing code to solve specific code
	Methodologies and Practices	Utilizes methodologies and Frameworks e.g. Agile to manage and streamline the development process.	Involves using specific paradigms e.g. procedural, object oriented, or functional programming
	Team and collaboration	Involves collaboration among diverse such as project managers, business analysts, developers, testers, and operations staff.	Can be done performed individually or in small teams with minimal formal structure
	Deliverables and Outcomes	Delivers comprehensive software solutions that are robust, user-friendly, and maintainable.	Delivers code that solves specific problem or implements specific features
	Quality and Testing	Emphasizes rigorous testing and quality assurance processes to ensure software reliability and performance	Testing is often performed by the developer writing code, focusing on immediate functionality.

Software Development Life Cycle (SDLC) refers to a structured and systematic approach to plan, develop, maintain, and enhance software with the goal of enhancing high-quality software that meets customer expectations and demands while minimizing project risks and cost.

Phases of software Development life cycle

1. **Planning:** Involves defining the project scope, objectives, and feasibility. It includes identifying the resources, budget, timeline and potential risks. Activities include;
    - *Conduct feasibility studies.*

- *Define project scope and objectives.*
- *Create a project plan and schedule.*
- *Identify resources and stakeholders.*
- *Risk assessment and mitigation planning.*

2. **Requirement Analysis:** Involves gathering detailed requirements from the stakeholders and documenting them. Activities include;
    - *Conduct stakeholder interviews and surveys.*
    - *Gather functional and non-functional requirements.*
    - *Create requirement specifications and use cases.*
    - *Validate and verify requirements with stakeholders.*

3. **System Design:** Involves creating the overall architecture and design of the system. It includes defining the system components, their interactions, and the user interface. Activities include
    - *Develop system architecture diagrams.*
    - *Design database schemas and data models.*
    - *Create detailed design documents for each component.*
    - *Design user interfaces and user experience (UI/UX).*

4. **Implementation (Coding) :** Involves the actual coding of the software based on the design documents. Activities include;
    - *Write code according to design specifications.*
    - *Perform unit testing on individual modules.*
    - *Integrate modules to build the complete system.*
    - *Conduct code reviews and peer reviews.*

5. **Testing :** Involves ensuring the software is free of defects and meets specified requirements. Testing performed include unit, integration, system, and acceptance testing. Activities include;
    - *Develop test plans and test cases.*
    - *Execute tests and document results.*
    - *Perform unit, integration, system, and user acceptance testing (UAT).*
    - *Identify and fix defects.*

6. **Deployment :** Involves deployment of the software to the production environment where it will be used by end-users. Activities include;
    - *Plan and execute the deployment process.*
    - *Configure production environments.*
    - *Perform final verification and validation.*
    - *Release software to end-users.*
    - *Provide user training and documentation.*

7. **Maintenance :** Involves ongoing support, bug fixes, updates, and enhancements to ensure the software continues to meet user needs. Activities include;
    - *Monitor software performance and usage.*
    - *Identify and fix bugs and issues.*
    - *Implement updates and enhancements.*
    - *Provide technical support and user assistance.*
    - *Conduct periodic reviews and optimizations.*

8. **Retirement :** Involves safely decommissioning the software, ensuring data preservation, and transitioning users to new systems if necessary. Activities include;
    - *Plan the decommissioning process.*
    - *Migrate data to new systems or archive it.*
    - *Notify stakeholders and users.*
    - *Remove software from production environments.*
    - *Conduct a post-mortem analysis to document lessons learned.*

Agile Model: refers to software development approach based on iterative and incremental development focusing on collaboration, customer feedback, and continuous improvement to deliver high-quality software effectively and efficiently.

Waterfall Model: refers to a linear software development approach that uses rigid phases i.e. when one phase ends, the next begins and if unmodified the model does not allow developers to go back to previous steps hence waterfall.

	Waterfall Model	**Agile Model**
Roles	Assigns roles to project team members, with specific duties and responsibilities defined for each team member.	Empowers team members to collaborate on different aspects of the project over time, leading to a more self-organizing team structure.
Planning	Planning is a linear process done at the beginning of the project, with all requirements and objectives laid out in detail upfront.	Planning is a continuous process throughout the project's life cycle, with adjustments made as new information or requirements emerge.
Scope	Discourages changes to the project's scope because the methodology requires an extensive amount of time spent in the beginning trying to get the plan right, which can make changes costlier after the project has begun.	Agile is more adaptable to changes in scope, with the development team able to adjust quickly as requirements change.
Time frames	Designed for long-term projects with predetermined timelines.	Uses short iterations to deliver value rapidly, allowing teams to adjust plans over time and achieve shorter time frames.
Speed	Waterfall projects tend to take longer because all requirements must be agreed upon before development can begin.	Agile projects, on the other hand, are usually delivered more rapidly than waterfall projects due to the iterative development cycles used in agile.
Delivery	Waterfall requires the completion of all tasks before any work can be released.	Agile allows for quick delivery of projects with shorter lifecycles, as each iteration delivers a workable product.
Flexibility	Waterfall is less flexible and resistant to change once the project's scope has been defined.	Agile encourages teams to respond quickly and adaptively to changes during the development process.

Testing	In waterfall, testing is usually done at specific milestones, often towards the end of the project.	Agile emphasizes incremental testing to identify and resolve issues throughout the development process.
Documentation	Waterfall, in contrast, relies heavily on documenting each step-in detail to ensure that all team members are on the same page.	Agile relies on minimal documentation, focusing on self-organizing teams and collaboration.
Communication	In waterfall, communication is more formal, with detailed communication plans and progress reports shared across multiple stakeholders.	Agile emphasizes informal communication, with frequent interactions between individuals or small groups of stakeholders.

Requirements Engineering: is the process of identifying, eliciting, analyzing, specifying, validating, and managing the needs and expectations of stakeholders for a software system.

The requirements engineering process involves several stages aimed at understanding, documenting, validating, and managing the needs for a software system. Here's a summary of each stage:

- **Feasibility Study:**

Technical Feasibility: Assesses the availability of required technology, resources, and the technical team's capabilities.

Operational Feasibility: Analyzes how well the software will meet user needs and how easy it will be to operate and maintain.

Economic Feasibility: Evaluates the cost-benefit aspect to determine if the project is financially viable.

Legal Feasibility: Reviews legal constraints and implications.

- **Requirements Elicitation:**

This involves gathering information about stakeholder needs and expectations using techniques like interviews, surveys, focus groups, observation, and prototyping. The goal is to understand the problem and broaden the domain knowledge of the analysts.

- **Requirements Specification:**

This stage produces formal models of the requirements, including functional and non-functional requirements and constraints. Common models include ER diagrams, DFDs, and FDDs. The result is a clear and comprehensive requirements document, which should be understandable by both the development team and stakeholders.

- **Requirements Verification and Validation (V&V):**

Verification: Ensures requirements are complete, consistent, and accurate.

Validation: Ensures the requirements meet stakeholder needs and expectations. Techniques include reviews, test case creation, and stakeholder walkthroughs. V&V is iterative and continues throughout the development life cycle.

- **Requirements Management:**

Involves tracking, documenting, prioritizing, and controlling changes to the requirements. Key activities include change tracking, version control, traceability, effective communication, and monitoring/reporting progress. This helps prevent scope creep and ensures alignment with project goals.

Advantages of Requirements Engineering Process:

- Helps ensure that the software being developed meets the needs and expectations of the stakeholders
- Can help identify potential issues or problems early in the development process, allowing for adjustments to be made before significant
- Helps ensure that the software is developed in a cost-effective and efficient manner
- Can improve communication and collaboration between the development team and stakeholders
- Helps to ensure that the software system meets the needs of all stakeholders.
- Provides an unambiguous description of the requirements, which helps to reduce misunderstandings and errors.
- Helps to identify potential conflicts and contradictions in the requirements, which can be resolved before the software development process begins.
- Helps to ensure that the software system is delivered on time, within budget, and to the required quality standards.
- Provides a solid foundation for the development process, which helps to reduce the risk of failure.

Modularity in software design refers to the practice of dividing a software system into distinct, independent units or modules, each responsible for a specific aspect of the system's functionality.

Modularity enhances the maintainability and scalability of software systems through the following mechanisms:

Maintainability

- Isolation of Changes: Changes in one module have minimal impact on others, reducing the risk of introducing bugs.
- Simplified Debugging and Testing: Individual modules can be tested and debugged separately, making it easier to identify and fix issues.
- Easier Understanding: Developers can focus on one module at a time, improving comprehension and efficiency.
- Enhanced Reusability: Well-defined modules can be reused across different projects, saving time and effort.
- Better Documentation: Clear definition of module purposes and interfaces improves overall documentation and understanding.

Scalability

- Independent Development and Deployment: Modules can be developed and deployed independently, speeding up the development process.
- Ease of Upgrades and Enhancements: New features can be added as new modules without affecting existing ones.
- Resource Management: Modules can be deployed on different servers, improving resource allocation and load balancing.
- Incremental Scaling: Systems can be scaled incrementally by adding more instances of specific modules.
- Improved Performance: Distributing modules across multiple servers enhances the system's ability to handle increased loads.

Software testing is the process of evaluating and verifying that a software application or system functions as intended. The primary goal of software testing is to identify defects, ensure quality, and validate that the software meets the specified requirements.

Levels of Software Testing

- Unit Testing:

Tests individual components or modules of a software to ensure they work correctly in isolation.

- Integration Testing:

Tests the interaction between integrated modules to identify interface defects.

- System Testing:

Tests the complete and integrated software to verify that it meets specified requirements.

- Acceptance Testing:

Conducted to determine whether the software system satisfies the business requirements and is ready for deployment. Types include User Acceptance Testing (UAT) and Operational Acceptance Testing (OAT).

Importance of Software Testing

- Ensures Software Quality:

Helps in delivering a quality product that meets customer expectations and requirements.

- Reduces Development Costs:

Identifies defects early in the development cycle, reducing the cost and effort required to fix them later.

- Enhances User Satisfaction:

Provides a reliable and robust product, leading to higher user satisfaction and trust.

- Facilitates Compliance and Standards:

Ensures that the software complies with industry standards, regulations, and guidelines.

- Improves Performance:

Identifies performance bottlenecks and ensures the software operates efficiently under various conditions**.**

Version Control Systems: is a tool that helps manage changes to source code or documents over time. It tracks the history of changes, allowing multiple developers to collaborate on a project simultaneously.

Benefits of version control systems

- Backup and Restore: Ensures no loss of work.
- Collaborative Development: Facilitates teamwork.
- History and Documentation: Provides detailed change records.
- Branching and Merging: Supports flexible development workflows.
- Improved Code Quality: Enables code reviews and continuous integration.

Popular Version Control Systems

- Git: Widely used DVCS with platforms like GitHub.
- Subversion (SVN): Popular CVCS for centralized control.
- Mercurial: DVCS similar to Git, designed for ease of use.
-

Software project management: is the process of planning, organizing, directing, and controlling resources to achieve specific software development goals within a set timeframe and budget.

A software project manager plays a pivotal role in overseeing and orchestrating the successful completion of software development projects.

Key aspects of their role include:

- Project Planning:

Defining the project scope, objectives, and deliverables.

Creating detailed project plans, including timelines, milestones, and resource allocation.

Estimating budgets and costs, and ensuring the project remains financially viable.

- Team Management:

Assembling and leading a team of developers, designers, testers, and other stakeholders.

Assigning tasks and responsibilities, ensuring each team member understands their role.

Facilitating communication and collaboration within the team.

- Risk Management:

Identifying potential risks and developing mitigation strategies.

Continuously monitoring risks throughout the project lifecycle and adjusting plans as needed.

- Resource Management:

Allocating and managing resources effectively to ensure project goals are met.

Balancing the needs of the project with available resources, including time, budget, and personnel.

- Quality Assurance:

Ensuring the software meets quality standards and requirements.

Overseeing testing processes to identify and address defects and issues.

- Stakeholder Communication:

Serving as the primary point of contact between the project team and stakeholders.

Regularly updating stakeholders on project progress, challenges, and successes.

Managing stakeholder expectations and incorporating feedback into the project.

- Problem Solving:

Addressing and resolving issues that arise during the project lifecycle.

Facilitating decision-making processes and ensuring timely resolution of problems.

- Performance Monitoring:

Tracking project progress against the plan using tools and metrics.

Conducting regular reviews and adjusting plans as necessary to stay on track.

Ensuring the project adheres to timelines and budget constraints.

- Project Closure:

Conducting final reviews and ensuring all project deliverables are completed.

Documenting the project and lessons learned for future reference.

Ensuring a smooth transition of the software product to maintenance or operational teams.

Some other challenges faced in managing software projects include:

Unclear goals

Misaligned goals and business objectives

Lack of accountability

Impractical deadlines

Poor risk management

Difficulty finding effective project management software

Software maintenance: is the process of modifying and updating software applications after their initial deployment to correct faults, improve performance, or adapt the software to a changed environment.

Maintenance is an essential part of the software lifecycle for several key reasons:

- Correcting Errors:

Despite rigorous testing, bugs and errors often emerge after deployment when the software is used in real-world conditions. Maintenance allows these issues to be identified and corrected, ensuring the software functions as intended.

- Adapting to Changing Environments:

Software needs to stay compatible with evolving hardware, operating systems, and third-party services. Adaptive maintenance ensures that the software continues to operate smoothly as the environment changes.

- Improving Performance and Functionality:

Over time, user needs and expectations evolve. Perfective maintenance allows for the addition of new features, performance enhancements, and overall improvements, keeping the software relevant and competitive.

- Ensuring Security:

New security vulnerabilities are constantly being discovered. Maintenance is crucial for applying security patches and updates to protect the software and its users from threats.

- Enhancing Reliability and Stability:

Preventive maintenance involves proactive measures to improve the software's reliability and prevent future issues. This includes code refactoring, updating documentation, and conducting regular audits.

- Prolonging Software Lifespan:

By continuously updating and refining the software, maintenance extends its useful life, delaying the need for a costly and time-consuming complete redevelopment.

- User Satisfaction:

Regular maintenance ensures that the software continues to meet user needs and expectations, leading to higher user satisfaction and retention.

- Cost Efficiency:

Addressing issues promptly through maintenance can prevent more significant problems and costly fixes later. It is generally more cost-effective to maintain and update existing software than to develop new software from scratch.

Software engineers face several ethical issues, including:

- Privacy and Data Protection: Responsible handling of user data and balancing security needs with privacy rights.
- Security: Addressing vulnerabilities responsibly and conducting ethical security assessments.
- Intellectual Property: Respecting copyright, licensing, and avoiding patent infringement.
- Professional Integrity: Maintaining software quality, providing accurate project information, and avoiding cutting corners.
- Algorithmic Fairness and Bias: Ensuring algorithms are fair, non-discriminatory, and transparent.
- User Consent and Autonomy: Obtaining informed consent and avoiding manipulative design practices.
- Impact on Society: Considering the societal and environmental impacts of software.
- Whistleblowing: Balancing the need to report misconduct with the risks involved.
- Conflict of Interest: Disclosing any potential conflicts of interest transparently.
- Access and Inclusion: Ensuring software is accessible and equitable for all users.

Software engineers can ensure adherence to ethical standards by:

- Educating Themselves: Familiarize with professional codes of ethics and stay informed on ethical issues.

- Ethical Design: Incorporate ethical considerations in the design phase, ensuring user consent and fairness.

- Workplace Culture: Model ethical behavior, participate in ethics committees, and conduct regular training.

- Data Privacy and Security: Implement strong data protection, access controls, and regular security audits.

- Transparency and Accountability: Be transparent about software functionality, maintain documentation, and establish feedback mechanisms.

- Stakeholder Engagement: Identify and consider the needs of all stakeholders, including marginalized groups.
- Legal Compliance: Ensure compliance with laws and ethical standards.
- Decision-Making Frameworks: Use ethical frameworks and scenario analysis for informed decisions.
- Professional Responsibility: Report unethical practices and seek peer reviews to maintain ethical standards.

Refferences:

- Radack, S. (2009). The system development life cycle (sdlc) (No. ITL Bulletin April 2009 (Withdrawn)). National Institute of Standards and Technology.
- Glass, R. L., Vessey, I., & Ramesh, V. (2002). Research in software engineering: an analysis of the literature. Information and Software technology, 44(8), 491-506.
- Biable, S. E., Garcia, N. M., Midekso, D., & Pombo, N. (2022). Ethical issues in software requirements engineering. *Software*, *1*(1), 31-52.
- Chatgpt