PLP SE-ASSIGNMENT-2

AFRICIA KERUBO NYAMBEGA

[keruboafricia@gmail.com](mailto:keruboafricia@gmail.com)

INTRODUTION TO SOFTWARE ENGINEERING

Define Software Engineering:

Software engineering is the discipline of applying engineering principles to the development of software, a systematic approach to design, build, test, deploy and maintain software systems.

What is software engineering, and how does it differ from traditional programming? Software Development Life Cycle (SDLC):

Software engineering is a discipline that encompasses the entire software development life cycle with a holistic approach that looks at the development process from initial conception to deployment and maintenance while traditional programming is focused on the act of writing code to solve specific problems.

Explain the various phases of the Software Development Life Cycle. Provide a brief description of each phase. Agile vs. Waterfall Models:

The Software Life Cycle is a framework that outlines the stages involved in building software, these are:

Planning and requirements gathering- This initial phase involves understanding and gathering the needs of the users and defining the functionality of the software.

Design-With a clear understanding of the requirements, software engineers translate them into technical specifications and system architecture for example flowcharts and data models.

Development- Developers code in a specific programming language based on the design documents.

Testing - Throughout the development process, rigorous testing is performed to identify and fix bugs using different testing techniques.

Deployment- Once tested and approved it is deployed to the production environment and deployed to users.

Maintenance-After deployment, the software is maintained by engineers who produce patches and updates to secure against vulnerabilities.

AGILE VS WATERFALL:

Waterfall is a traditional model that’s linear where requirements are gathered upfront and the project progresses through the next stages in a fixed order, changes are difficult to implement while Agile model is an iterative and incremental model which emphasizes flexibility and adaptation. Projects are broken down into sprints with continuous development and testing which allows for adaptation.

The choice between agile and waterfall depends on the specific project requirements, where evolving requirements are suited for agile model while projects with well-defined requirements are less dynamic in nature.

Compare and contrast the Agile and Waterfall models of software development. What are the key differences, and in what scenarios might each be preferred? Requirements Engineering:

Structure: Agile is iterative and incremental, with frequent delivery of working software in short cycles (sprints). Waterfall is linear and sequential, progressing through SDLC phases only once.

Flexibility: Agile is highly adaptable, allowing for changes in requirements throughout the project. Waterfall is less flexible, with changes after the initial planning phase being complex and costly.

Documentation: Agile focuses on lightweight documentation and prioritizes working code and collaboration. Waterfall involves more detailed documentation at each stage.

Customer Involvement: Agile emphasizes ongoing customer interaction and feedback throughout the development process. Waterfall has less frequent customer interaction, primarily in the initial stages.

Ideal Use Cases:

Agile: When requirements are unclear or likely to change, when projects are complex and user feedback is crucial, or when rapid delivery of features is a priority. Agile is well-suited for modern web applications, mobile apps, and projects with a high degree of uncertainty.

Waterfall: When requirements are well-defined and stable, when projects have clear deadlines and a fixed budget, or when working with stricter regulations or compliance needs. Waterfall might be a good choice for infrastructure projects, system integrations, or projects with well-established requirements.

What is requirements engineering? Describe the process and its importance in the software development lifecycle. Software Design Principles:

Requirements Engineering is the systematic process of defining, documenting, and managing the needs of all stakeholders that is users, customers for a software system.

Process

Requirements Elicitation- gathering requirements from various stakeholders using different techniques.

Requirements Analysis- requirements are analyzed for clarity, consistency, completeness. It may involve prioritizing requirements.

Requirements Specification-Clear and concise documents are created that detail functional and non-functional requirements.

Requirements Validation-Stakeholders review the document for accuracy

Requirements Management- requirements are tracked and managed throughout the development lifecycle.

Importance

Reduced risk of failure

Improved communication

Better resource allocation

Efficient development

Software design principles

Modularity: Breaking down the software into smaller, independent modules that can be developed, tested, and maintained separately.

Abstraction: Focusing on the essential details and hiding unnecessary complexity from the user.

Encapsulation: Bundling data and the code that operates on that data together to protect data integrity and promote modularity.

Separation of Concerns: Designing different parts of the software to handle specific functionalities independently.

Reusability: Creating code components that can be reused in different parts of the software or even in future projects.

Explain the concept of modularity in software design. How does it improve maintainability and scalability of software systems?

Modularity in software design is the principle of dividing a software program into smaller, self-contained units called modules that perform specific tasks and interact with each other through interfaces.

How does it improve maintainability and Scalability:

Isolation of changes within specific modules

Improved code readability

Improved testability as individual modules can be tested in isolation simplifying the testing process.

Modular reuse as modules can be reused in different parts of the same software

Describe the different levels of software testing (unit testing, integration testing, system testing, acceptance testing). Why is testing crucial in software development? Version Control Systems:

1. Unit Testing:

Focus: Individual software units (modules, functions, classes) are tested in isolation to verify they behave as expected according to their design specifications.

Methodology: Unit tests are typically written by developers using unit testing frameworks. These tests are automated and can be run frequently throughout the development process.

2. Integration Testing:

Focus: This level tests how different software units interact and work together as a whole. It verifies data flow and communication between integrated modules.

Methodology: Integration tests can involve combining multiple modules and testing their interactions. Stubs and drivers (simulated modules) may be used to isolate specific parts of the system during testing.

3. System Testing:

Focus: The entire software system is tested as a whole against the system requirements. This includes functional testing (verifying features work as intended) and non-functional testing (performance, usability, security).

Methodology: System testing often involves a combination of automated and manual testing techniques.

4. Acceptance Testing:

Focus: This final level of testing is performed by the end-users or customers to ensure the software meets their acceptance criteria and business needs.

Methodology: Acceptance testing can involve user acceptance testing (UAT) where users provide feedback and identify any usability issues. There can also be alpha and beta testing where the software is released to a limited group of users for broader feedback.

Why Testing is Crucial:

Early Bug Detection

Improved Quality

Reduced Risk

Enhanced User Experience

What are version control systems, and why are they important in software development? Give examples of popular version control systems and their features. Software Project Management:

Version control systems (VCS) are software tools that act as a central repository for all versions of your code, tracking changes over time. They're vital for managing collaborative software projects, ensuring smooth development workflows, and enabling features like:

Version History: Track every modification made to the codebase, allowing you to revert to previous versions if necessary.

Collaboration: Enable multiple developers to work on the same project simultaneously, merging changes seamlessly to avoid conflicts.

Branching and Merging: Create isolated branches for experimentation or feature development, then easily integrate them back into the main codebase.

Code Rollback: Accidentally deleted or broke something? No problem! Easily revert to a stable version.

Parallel Development: Teams can work on different parts of the codebase independently without stepping on each other's toes.

Improved Communication: Version control systems provide a clear audit trail of changes, making communication and code reviews more efficient.

Popular VCS:

Git

Mercurial

Effective project management is crucial for ensuring successful software development its key aspects are:

Project Planning: Defining project scope, setting realistic milestones, and estimating resource requirements (time, budget, personnel).

Task Management: Breaking down project work into manageable tasks, assigning them to team members, and tracking progress.

Risk Management: Identifying potential risks that could impact the project and developing mitigation strategies.

Communication Management: Ensuring clear and frequent communication between all stakeholders (developers, clients, managers) throughout the project lifecycle.

Issue Tracking: Identifying, documenting, and resolving bugs, defects, and other issues that arise during development.

Version Control Integration: Version control systems play a vital role in software project management by enabling features like version history, branching for parallel development, and improved team collaboration

Discuss the role of a software project manager. What are some key responsibilities and challenges faced in managing software projects? Software Maintenance:

The software project manager plays a pivotal role in the success of any software development project. They act as the glue that binds various elements together, ensuring a smooth and efficient development process.

Responsibilities:

Project Planning and Scope Definition: Working with stakeholders to define project goals, features, functionality, and time-lines.

Resource Management: Assigning tasks, managing personnel, and allocating resources (time, budget, tools) effectively to meet project goals.

Risk Management: Identifying potential risks that could derail the project and developing mitigation plans to address them proactively.

Communication Facilitation: Ensuring clear and transparent communication between all stakeholders throughout the project lifecycle.

Progress Monitoring and Reporting: Tracking project progress, identifying roadblocks, and reporting on key metrics to stakeholders.

Vendor Management: If the project involves external vendors or contractors, the SPM oversees their work, manages communication, and ensures they meet their deliverables.

Challenges:

Balancing Scope, Time, and Budget: The project management triangle – managing scope, time, and budget – is a constant challenge.

Managing Stakeholder Expectations: Stakeholders may have varying priorities and expectations.

Technical Complexity: Software development projects can involve complex technologies. While SPMs may not be technical experts themselves, they need a good understanding of the technical landscape to make informed decisions and guide the project effectively.

Team Dynamics and Motivation: Leading and motivating a team of developers, designers, and testers requires strong interpersonal and leadership skills.

Define software maintenance and explain the different types of maintenance activities. Why is maintenance an essential part of the software lifecycle? Ethical Considerations in Software Engineering:

Software maintenance is the continual process of modifying and updating software after it's been deployed.

· Corrective Maintenance: This involves fixing bugs and defects in the software that cause errors or unexpected behavior. These issues can be reported by users, identified during testing, or discovered through monitoring the software in production.

· Adaptive Maintenance: This type of maintenance focuses on adapting the software to changes in the external environment. This could involve:

· Compatibility Updates: Ensuring the software works with new operating systems, hardware, or other software versions.

Regulatory Compliance: Adapting the software to meet new legal or regulatory requirements.

· Perfective Maintenance: This proactive approach focuses on improving the software's functionality, performance, usability, or security. It may involve:

· Performance Optimization: Identifying and addressing bottlenecks to improve the software's speed and responsiveness.

Adding New Features: Implementing new functionalities based on user feedback or changing market demands.

· Preventive Maintenance: This forward-thinking approach involves making changes to the software's code to prevent future problems. This could include:

· Code Refactoring: Improving the code's structure and readability to make it easier to maintain and understand in the long run.

What are some ethical issues that software engineers might face? How can software engineers ensure they adhere to ethical standards in their work?

Privacy and Security:

· Data Collection and Use: Software engineers need to consider how much user data is collected, how it's stored, and who has access to it.

. Security Vulnerabilities: Unintentional security vulnerabilities can leave user data exposed to breaches. Engineers have a responsibility to write secure code and address vulnerabilities promptly. The 2017 Equifax data breach, where a software vulnerability exposed millions of users' personal information, is a stark reminder of the consequences of lax security practices.

· Bias and Fairness: Algorithms can perpetuate societal biases if not carefully designed and tested. For example, an algorithm used in a loan application process could unfairly discriminate against certain demographics based on historical data.

· Accessibility: Not everyone has the same abilities. Software engineers should strive to create inclusive software that caters to users with disabilities.

· Environmental Impact: The software development lifecycle has an environmental footprint. Energy consumption during operation and the materials used in hardware all have environmental consequences. Software engineers can consider energy-efficient coding practices and advocate for sustainable practices within their organizations.

**References:**

[1] ACM Code of Ethics and Professional Conduct: [https://www.acm.org/code-of-ethics](https://www.acm.org/code-of-ethics" \t "_blank)