**Software Engineering**

**Definition**: The application of engineering principles and techniques to software development. Software engineering comprises methods, practices, and tools for the engineering of software.

**Difference from Traditional Programming**:

1. **Scope**: Traditional programming is about coding to address a particular set of problems or tasks. Software engineering has a wider scope which covers analysis, design, implementation, testing, maintenance and evaluation of a project.
2. **Methodologies**: Software engineering utilizes defined methodologies and best practices to ensure quality and maintainability of the software product. Traditional programming may or may not involve these engineering processes.
3. **Collaboration**: Software engineering usually requires collaboration and coordination with different project stakeholders (e.g., developer, tester, manager) while traditional programming is an individual effort.

**Software Development Life Cycle (SDLC)**

**Phases**:

1. **Planning**:

* **Description**: Identify project scope, objectives and feasibility. Set project goals and limitations.
* **Outcome**: Project plan, resources and schedule.

1. **Requirements Analysis**:

* **Description**: Collect and analyze project requirements from stakeholders. Write functional and non-functional requirements.
* **Outcome**: Requirement’s document, functional specifications.

1. **Design**:

* **Description**: Develop software architecture and detailed design. Specify components, interfaces and data flow.
* **Outcome**: Design documentation, UML diagrams.

1. **Implementation (Coding)**:

* **Description**: Develop executable software code from design specifications using relevant programming languages.
* **Outcome**: Source code.

1. **Testing**:

* **Description**: Evaluate software against user requirements. Detect and resolve defects.
* **Outcome**: Tested and validated software.

1. **Deployment**:

* **Description**: Install and setup software in production environment. Conduct user training and support.
* **Outcome**: Deployed software, user guides.

1. **Maintenance**:
   * **Description**: Provide bug fixes, improvements and support to existing software. Ensure software stability and relevance.
   * **Outcome**: Maintained software.

**Agile vs. Waterfall Models**

**Agile**:

1. **Description**: A software development approach based on iterative and incremental progress. Emphasizes on flexibility, team collaboration and customer feedback.
2. **Key Features**: Brief development cycles (called sprints), continuous improvement, adaptive planning.
3. **Preferred Scenarios**: When requirements are not fully known, need for speedy software delivery and tight collaboration with the customer.

**Waterfall**:

1. **Description**: A linear and sequential software development paradigm. Each development phase must be completed in one go before proceeding to the next phase.
2. **Key Features**: Well defined and structured phases, detailed documentation and milestones.
3. **Preferred Scenarios**: Projects with documented requirements, minimum scope change and regulatory compliances.

**Comparison**:

1. **Flexibility**: Agile offers high flexibility and adaptability to changes while Waterfall is inflexible.
2. **Customer Involvement**: In Agile, customer feedback is considered throughout the development lifecycle; whereas in Waterfall it’s only at the beginning and end.

**Requirements Engineering**

**Definition**: The process of establishing and documenting the requirements of a software system.

**Process**:

1. **Requirements Elicitation**: Elicit requirements from stakeholders using interviews, surveys and observations.
2. **Requirements Analysis**: Analyse and prioritize requirements to make them unambiguous, complete and feasible.
3. **Requirements Specification**: Specifying the requirements in a detailed and structured document.
4. **Requirements Validation**: Validate the requirements specification with stakeholders to ensure that the documented requirements meet their needs and expectations.
5. **Requirements Management**: Manage requirements throughout the project lifecycle by tracking changes and ensuring that they are properly documented.

**Importance**:

1. To ensure that the ultimate software product satisfies user needs and expectations.
2. To identify potential problems as early as possible in the development process.
3. To facilitate clear communication between stakeholders.

**Software Design Principles**

**Modularity**:

1. **Concept**: Breaking a software system into smaller, manageable and independent units or modules.
2. **Improvement in Maintainability**: Individual modules are easier to update, debug and modify without affecting the overall system.
3. **Improvement in Scalability**: Each module can be developed, tested and deployed independent of other modules. Thus, allowing scalable growth.

**Testing in Software Engineering**

**Levels of Software Testing**:

1. Unit Testing:

* Description: Testing of individual components or functions to ensure they function correctly as per the definition.
* Importance: To detect bugs as early as possible in the development cycle.

1. Integration Testing:

* Description: Testing of integrated components to reveal interface defects.
* Importance: To ensure that component parts work well together.

1. System Testing:

* Description: Testing of a system as a whole to prove that it fulfils its specified requirements.
* Importance: To validate the overall functionality and performance of the system.

1. Acceptance Testing:

* Description: Testing done to verify whether the system is acceptable to the end-user or customer and is ready to be released.
* Importance: System acceptance to the end-user or customer.

**Importance of Testing**:

1. Maintains the quality and reliability of the software.
2. Detects and corrects bugs before the software is delivered.
3. Increases user satisfaction and confidence.

**Version Control Systems**

**Definition**: A version control system (VCS) is a tool that keeps track of changes in source code over time.

**Importance**:

1. Allows multiple developers to collaborate easily.
2. Tracks every change in the codebase.
3. Rolls back to a prior version if a mistake is detected. **Examples**:
   1. **Git:** 
      1. Features: Distributed VCS, branching and merging, and strong community.
      2. Popular Platforms: GitHub, GitLab, Bitbucket.
   2. **Subversion (SVN):** 
      1. Features: Centralized VCS, directory versioning, and atomic transactions.
      2. Popular Platforms: Apache Subversion.

**Software Project Management**

**Software Project Manager Role**:

1. Planning, overseeing, and terminating projects.
2. Managing resources, time, cost, and risk factors.
3. Delivering projects with quality guidelines. **Challenges**:
4. Coping with changing requirements.
5. Deadlines.
6. Limited resources.
7. Communication with stakeholders.

**Software Maintenance**

**Definition**: Any modification and extension of a software product after it has been released to users in order to correct it, make it more efficient or to accommodate new requirements.

**Types of Maintenance**:

1. **Corrective Maintenance**: Repairing faults.
2. **Adaptive Maintenance**: Making software compatible with different environment (e.g., OS).
3. **Perfective Maintenance**: Improving the existing functions and performance.
4. **Preventive Maintenance**: Detecting and removing potential faults.

**Importance**:

1. Keeps the software in a proper working condition and performance.
2. Meets the changing needs of users and technological advancement.
3. Prolongs the life of the software product and keeps users’ satisfaction.

**Ethical Considerations in Software Engineering**

**Ethical Issues**:

1. **Privacy**: Protect user’s privacy and personal information.
2. **Security**: Create secure code and protect against outside access.
3. **Intellectual Property**: Not infringe any patents.
4. **Transparency**: Does not hide truth about functionality and capability.

**Compliance with ethical standards**:

1. Adhere to industry codes of conduct and professional guidelines.
2. Apply best practices to privacy and security.
3. Be transparent with all stakeholders about the project as well as any problems or delays.
4. Periodically review and update ethical guidelines.

**References**

Sommerville, I. (2016). Software engineering (10th ed.). Pearson.

Brooks, F. P., Jr. (1995). The mythical man-month: Essays on software engineering (Anniversary ed.). Addison-Wesley.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). Design patterns: Elements of reusable object-oriented software. Addison-Wesley.

Martin, R. C. (2002). Agile software development, principles, patterns, and practices. Prentice Hall.

DeMarco, T. (1997). The deadline: A novel about project management. Dorset House Publishing.

George, M. L., Maxey, J., Rowlands, D., & Upton, M. (2005). The lean six sigma pocket toolbook: A quick reference guide to 100 tools for improving quality and speed. McGraw-Hill.