Principles of Software Engineering I

Module 1: Software Engineering

Lesson 1: Software

Overview

Software is a set of instructions, data, or programs used to operate computers and execute specific tasks. It can be categorized as system software, application software, or development tools.

Key Concepts

- **Types of Software**:
  - **System Software**: Operating systems, device drivers (e.g., Windows, Linux).
  - **Application Software**: End-user applications (e.g., web browsers, word processors).
  - **Development Tools**: Compilers, IDEs (e.g., Visual Studio Code).
- **Characteristics**:
  - Intangible: Cannot be physically touched.
  - Evolvable: Can be updated or modified.
  - Complex: Often involves thousands of lines of code.

Practical Example

- **Analyze Software Types**:
  - Identify software on your computer (e.g., Chrome as application software, Windows as system software).
  - Discuss how each serves a different purpose.

Lesson 2: Engineering

Overview

Engineering applies scientific principles to design and build systems. Software engineering adapts these principles to create reliable, efficient, and scalable software.

Key Concepts

- **Software Engineering vs. Other Engineering**:
    - Similarities: Systematic design, problem-solving, testing.
    - Differences: Software is intangible, evolves rapidly, and faces unique challenges like changing requirements.
- **Challenges**:
    - Complexity: Managing large codebases.
    - Scalability: Ensuring performance under load.
    - Maintainability: Updating software without introducing errors.

Practical Example

- **Compare Disciplines**:
    - Civil engineering builds bridges with fixed requirements; software engineering builds applications with evolving requirements.
    - Example: A bridge's design is static; a web app's features may change based on user feedback.

Lesson 3: Development Process

Overview

The software development process is a structured approach to building software, involving planning, design, implementation, testing, and maintenance.

Key Concepts

- **Process Goals**:
    - Deliver functional software.
    - Meet user requirements.
    - Ensure quality and reliability.
- **Key Phases**:
    - Planning: Define scope and resources.
    - Development: Write and test code.
    - Deployment: Release software to users.
    - Maintenance: Fix bugs and add features.

Module 2: Software Development Lifecycle (SDLC)

Lesson 1: Lifecycle Overview & Requirements

Overview

The SDLC is a framework defining tasks performed at each step of software development. Requirements gathering is the first step, identifying what the software must do.

Key Concepts

- **SDLC Phases**:
  - Requirements: Gather user needs.
  - Design: Plan the solution.
  - Implementation: Build the software.
  - Testing: Verify functionality.
  - Maintenance: Update and fix issues.
- **Requirements Types**:
  - Functional: Features (e.g., user login).
  - Non-functional: Performance, usability, security.

Practical Example

- **Gather Requirements**:
  - For a library management system:
    - Functional: Borrow/return books, search catalog.
    - Non-functional: Support 100 concurrent users, load in under 2 seconds.

Lesson 2: Specification

Overview

Specifications translate requirements into detailed plans, describing how the software will function and meet user needs.

Key Concepts

- **Purpose**: Guides developers and ensures alignment with user expectations.
- **Components**:
  - Functional Specifications: Detailed feature descriptions.
  - Technical Specifications: System architecture, technologies used.
- **Tools**: Use cases, user stories, flowcharts.

Practical Example

- **Write a Specification**:
    - For a note-taking app:
        - User Story: “As a user, I want to save notes so I can access them later.”
        - Technical Spec: Use SQLite for local storage, REST API for cloud sync.

Lesson 3: Design & Implementation

Overview

Design creates the blueprint for the software, while implementation involves writing the code based on the design.

Key Concepts

- **Design Principles**:
    - Modularity: Break system into components.
    - Scalability: Plan for growth.
    - Reusability: Use components across projects.
- **Implementation**:
    - Write code using chosen languages (e.g., Python, Java).
    - Follow coding standards (e.g., PEP 8 for Python).

Practical Example

- **Design a System**:
    - For a note-taking app, design a modular architecture:
        - UI Module: React for front-end.
        - Backend Module: Node.js with Express.
        - Database Module: MongoDB.
- **Implement a Feature**:
    - Code a simple note-saving function in Python:

    ```
    def save_note(title, content):
        with open("notes.txt", "a") as file:
            file.write(f"{title}: {content}\n")
    ```

Lesson 4: Testing & Maintenance

Overview

Testing verifies that the software meets requirements, while maintenance ensures it remains functional and relevant.

Key Concepts

- **Testing Types**:
  - Unit Testing: Test individual components.
  - Integration Testing: Test combined components.
  - System Testing: Test the entire system.
  - Acceptance Testing: Validate against user requirements.
- **Maintenance**:
  - Corrective: Fix bugs.
  - Adaptive: Update for new environments.
  - Perfective: Add new features.

Practical Example

- **Write a Test**:
  - Unit test for the note-saving function using Python's `unittest`:

    ```
    import unittest
    def save_note(title, content):
        with open("notes.txt", "a") as file:
            file.write(f"{title}: {content}\n")
    class TestNoteApp(unittest.TestCase):
        def test_save_note(self):
            save_note("Test", "This is a test note")
            with open("notes.txt", "r") as file:
                self.assertIn("Test: This is a test note", file.read())
    ```

Module 3: Lifecycle Models and Processes

Lesson 1: Lifecycle Models

Overview

Lifecycle models define the sequence and structure of SDLC phases. Common models include Waterfall, Iterative, and Agile.

Key Concepts

- **Waterfall**:
  - Linear, sequential phases.
  - Best for projects with fixed requirements.
- **Iterative**:
  - Develop in cycles, refining each iteration.
  - Suitable for evolving requirements.
- **Agile**:
  - Incremental, flexible, user-focused.
  - Ideal for dynamic projects.

Lesson 2: Agile and Scrum

Overview

Agile is a flexible, iterative approach emphasizing collaboration and customer feedback. Scrum is a popular Agile framework.

Key Concepts

- **Agile Principles**:
  - Deliver working software frequently.
  - Welcome changing requirements.
  - Collaborate closely with customers.
- **Scrum Components**:
  - Roles: Product Owner, Scrum Master, Development Team.
  - Ceremonies: Sprint Planning, Daily Scrum, Sprint Review, Sprint Retrospective.
  - Artifacts: Product Backlog, Sprint Backlog, Increment.

Practical Example

- **Run a Scrum Sprint**:
  - Product Backlog: List of features (e.g., "Add note search").
  - Sprint Planning: Select tasks for a 2-week sprint.
  - Daily Scrum: 15-minute stand-up to discuss progress.
  - Sprint Review: Demo completed features.

Module 4: The Project Team

Lesson 1: Product/Project Manager

Overview

The Product/Project Manager defines the product vision, prioritizes features, and ensures project delivery.

Key Concepts

- **Responsibilities**:
  - Define requirements and prioritize backlog.
  - Communicate with stakeholders.
  - Manage timelines and resources.
- **Skills**:
  - Leadership, communication, problem-solving.

Practical Example

- **Prioritize a Backlog**:
  - For a note-taking app, rank features: note search, cloud sync, offline mode.

Lesson 2: UX Designer, Engineer/Architects

Overview

UX designers focus on user experience, while engineers and architects build and design the technical solution.

Key Concepts

- **UX Designer**:
  - Creates wireframes, prototypes, and user flows.
  - Ensures usability and accessibility.
- **Engineer/Architect**:
  - Designs system architecture (e.g., microservices).
  - Writes and tests code.
  - Ensures scalability and performance.

Practical Example

- **UX Design**:
  - Create a wireframe for a note-taking app's main screen (use tools like Figma or sketch by hand).
- **Architecture Design**:
  - Propose a client-server architecture for the app:
    - Client: React front-end.
    - Server: Node.js with MongoDB.

Learning Outcomes Recap

By completing this course, students will:

1. Define software engineering and its unique challenges.
2. Describe SDLC phases and their importance.
3. Compare lifecycle models (Waterfall, Iterative, Agile).
4. Explain the role of design and specification in development.
5. Understand testing methods and maintenance strategies.
6. Identify project team roles and their contributions.
7. Apply software engineering principles to real-world problems.