SEMESTER-V

COURSE 11: SOFTWARE ENGINEERING

Theory **Credits: 3** **3 hrs/week**

Course Objectives:

1. Understand the fundamental principles of software engineering, including software development life cycle models and their practical applications.
2. Analyze and document software requirements through feasibility studies and specification techniques.
3. Apply software design principles and development standards for building reliable and maintainable systems.
4. Evaluate software testing methodologies, including design and execution of test cases for various testing levels.
5. Examine software maintenance strategies, types, and metrics to sustain software performance over time.

Course Outcomes

At the end of the course, students will be able to:

1. Compare and contrast software development life cycle models such as Waterfall, Spiral, and Agile, and explain their appropriate use cases.
2. Conduct requirement analysis and distinguish between functional and non-functional requirements to develop a Software Requirements Specification (SRS) document.
3. Utilize design principles like modularity, cohesion, and coupling; implement Data Flow Diagrams (DFDs), structure charts, and follow coding standards and version control practices.
4. Design and perform different types of software tests—white-box, black-box, unit, integration, system—and differentiate between manual and automated testing approaches.
5. Categorize software maintenance types and propose strategies based on software maintenance metrics for effective long-term software sustainability.

Unit 1: Software Engineering Foundations and Requirements Engineering:
Definition of Software engineering, Software life cycle models: Waterfall, prototyping, Evolutionary, Spiral and Agile models. Comparison among development life cycles.

Unit 2: Requirement Analysis and Specification
Feasibility study, Requirements gathering, Functional and Non-functional requirements, Requirements analysis and specification, design of software requirement specification (SRS).

Unit 3: Software Design Principles and Development Practices
Introduction to software design, modularity, cohesion, coupling and layering, functional design, and solution design, use of DFD and Structure chart in software design, UML in software design, user interface design. Software Development Basics, Coding standards, Version Control and Code review techniques.

Unit 4: Software Testing
Fundamentals of testing (verification and validation), White-box, and black-box testing, unit testing, integration testing, system testing, acceptance testing (alpha testing and beta testing), test scenarios and test case design, automation testing and manual testing.

Unit 5: Software Maintenance
Introduction to software maintenance, corrective maintenance, perfective maintenance, adaptive maintenance, preventive maintenance, challenges in software maintenance, metrics related to software maintenance.

Textbooks:

1. Software Engineering: A Practitioner's Approach, Roger Pressman, McGraw Hill, 6th Edition
2. Software Engineering, Sommerville, Adison Wesley, 7th edition.

Reference Books:

1. Fundamentals of Software Engineering, Mall Rajib,PHI, Fifth Edition,
2. Fundamentals of Software Engineering, Hitesh, BPB Publications

Activities:

Outcome: Compare and contrast software development life cycle models such as Waterfall, Spiral, and Agile, and explain their appropriate use cases.

Activity: Create a comparison chart in groups showing key features, pros/cons, and use cases of Waterfall, Spiral, and Agile models. Include a real-world example for each.

Evaluation Method: Use a rubric to assess on a 10-point scale to check:

- Accuracy of model descriptions
- Clarity of comparison
- Relevance of examples
- Presentation skills (if shared in class)

Outcome: Conduct requirement analysis and distinguish between functional and non-functional requirements to develop a Software Requirements Specification (SRS) document.

Activity: Analyze a simple system (e.g., online library or food ordering app). Identify 5 functional and 3 non-functional requirements. Draft a mini-SRS document using a template.

Evaluation Method: Checklist-based review on a 10-point scale:

- Correct classification of requirements
- Completeness of SRS sections
- Clarity and formatting
- Use of standard terminology

Outcome: Utilize design principles like modularity, cohesion, and coupling; implement Data Flow Diagrams (DFDs), structure charts, and follow coding standards and version control practices.

Activity: Design a basic login system using DFD (Level 0 and Level 1) and a structure chart. Highlight modules and discuss cohesion and coupling in pairs.

Evaluation Method: Peer review and instructor feedback on a 10-point scale:

- Correct use of DFD symbols
- Logical flow and modularity
- Explanation of cohesion/coupling
- Neatness and labelling

Outcome: Design and perform different types of software tests—white-box, black-box, unit, integration, system—and differentiate between manual and automated testing approaches.

Activity: Write simple test cases for a calculator app (e.g., addition, division by zero). Perform manual unit testing and simulate black-box and white-box testing.

Evaluation Method: Observation and worksheet to assess on a 10-point scale:

- Correct identification of test types
- Clear test case steps and expected output
- Execution and result recording
- Understanding of manual vs automated testing

Outcome: Categorize software maintenance types and propose strategies based on software maintenance metrics for effective long-term software sustainability.

Activity: Categorize sample maintenance tasks (e.g., fixing a bug, adding a feature) into corrective, adaptive, perfective, or preventive. Discuss sustainability strategies in small groups.

Evaluation Method: Group activity score (10-point scale):

- Correct classification of maintenance types
- Participation in discussion
- Suggestions for sustainability (e.g., documentation, modular design)
- Use of basic metrics (e.g., number of bugs fixed)

SEMESTER-V

COURSE 11: SOFTWARE ENGINEERING

Practical **Credits: 1** **2 hrs/week**

Select domain of interest (e.g. College Management System) and identify multi-tier software applications to work on (e.g. Online Fee Collection). Analyze, design and develop this application:

1. Develop an IEEE standard SRS document. Also develop risk management and project plan (Gantt chart).
2. Understanding of System modeling: Data model i.e. ER – Diagram and draw the ER Diagram with generalization, specialization and aggregation of specified problem statements.
3. Understanding of System modeling: Functional modeling: DFD Context and draw it
4. Understanding of System modeling: UML and draw it.
5. Develop a sample calculator program and perform Black-box Testing:
    a. Identify test scenarios without viewing the internal code.
    b. Write test cases for valid and invalid inputs.
    c. Execute the test cases and record outcomes.
6. Develop a sample login authentication page and perform White--box Testing**:**
    a. Analyze the code for all possible paths, conditions, and loops.
    b. Apply statement coverage and branch coverage techniques.
    c. Test internal functions with known inputs.
7. For the above login authentication page, perform the following:
    a. Simulate the following maintenance activities:
        - **Corrective Maintenance**: Fix an existing bug (e.g., wrong output, crash, miscalculation).
        - **Perfective Maintenance**: Add a new user-requested feature (e.g., sorting, filter, or improved UI).
        - **Adaptive Maintenance**: Modify the code to adapt to a new platform or library version.
        - **Preventive Maintenance**: Refactor the code to improve readability, performance, or security.
    b. Document each change with:
        - Problem description
        - Type of maintenance
        - Before and after screenshots
        - Change summary