UNIT-I

1a) Software engineering applies engineering principles to develop, test, and maintain software. Types include system, application, embedded, web, and AI-based software.

1b) Software process models: Waterfall (sequential), Iterative (incremental), V-Model (testing integration), Agile (flexible), Spiral (risk management), RAD (rapid development).

2a) Software process structures development; myths are misconceptions, like "adding programmers speeds work," ignoring onboarding delays, or "delivered software needs no maintenance."

2b) Prescriptive models have structured phases, predictability, risk management, and quality assurance. Examples: Waterfall, V-Model. They ensure discipline but lack flexibility.

UNIT-II

3a) SRS defines system requirements, functionalities, and constraints. It includes functional, non-functional, and design constraints, ensuring clarity and reducing misunderstandings.

3b) Cohesion measures module relatedness; high cohesion improves maintainability. Coupling measures module dependency; low coupling enhances flexibility and debugging.

4a) Requirements gathering involves identifying user needs via interviews, surveys, and document reviews. Proper analysis prevents ambiguities, ensuring clear software goals.

4b) Design transforms requirements into architecture. High-level design structures systems; low-level details components. Principles ensure scalability, maintainability, and efficiency.

UNIT-III

5a) Function-oriented design divides systems into modules. Features: modular decomposition, process-driven structure, hierarchy, and strong functional relations. Uses DFDs for visualization.

5b) UI types: GUI (Windows), CLI (Linux terminal), VUI (Alexa), Touch-based (smartphones). A good UI enhances usability, accessibility, and experience.

6a) A DFD represents data movement. Entities: customers, staff. Processes: booking, billing. Data stores: customer details, reservations. Levels detail processes.

6b) OOD structures software using objects/classes. Features: encapsulation, inheritance, polymorphism, abstraction. Benefits: reusability, maintainability, scalability, and modularization.

UNIT-IV

7a) Coding implements specifications; code review identifies errors, ensures readability, and enhances security, reducing debugging efforts and improving software quality.

7b) Testing verifies software. Types: Unit (components), Integration (modules), System (full validation), Acceptance (user approval), Black-box, White-box, and Performance testing.

8a) Black-box testing assesses functionality without code access; white-box tests internal logic. Black-box is user-centric; white-box is developer-centric.

8b) Object-oriented testing covers unit (classes), integration (interactions), system (functionality), and regression (modifications), ensuring reliable, maintainable, and efficient software.

UNIT-V

9a) Reliability ensures failure-free operation; availability measures uptime percentage. Reliability reduces crashes, while availability ensures software remains accessible.

9b) Software Quality Management ensures standards via Quality Assurance (preventive), Quality Control (defect detection), Testing (verification). Frameworks: ISO 9001, CMMI.

10a) Maintenance models: Corrective (bug fixes), Adaptive (environmental changes), Perfective (performance enhancements), Preventive (future failure prevention). Ensures software longevity.

10b) SQMS ensures software quality via Six Sigma, CMMI, ISO 9001. Focuses on defect prevention, process improvement, and customer satisfaction.