Unit 1: Introduction

1. Define Software Engineering. Discuss the Key Challenges Faced by Software Engineers.

Definition:
Software engineering is the systematic application of engineering principles to the design, development, testing, deployment, and maintenance of software systems. It involves using structured methodologies, tools, and techniques to ensure the creation of high-quality, reliable, and maintainable software.

Key Challenges:

1. **Managing Complexity**:
    - Software systems are inherently complex due to their size, functionality, and interactions with other systems.
    - Example: Developing an operating system like Windows involves managing millions of lines of code and ensuring compatibility with various hardware and software.
2. **Changing Requirements**:
    - Requirements often change during development due to evolving user needs or market conditions.
    - Example: A social media app may need to add new features (e.g., stories, reels) to stay competitive.
3. **Time and Budget Constraints**:
    - Projects often face tight deadlines and limited budgets, leading to compromises in quality.
    - Example: A startup may rush to release a minimum viable product (MVP) to attract investors.
4. **Ensuring Quality**:

- Delivering bug-free software that meets user expectations is challenging.
- Example: A banking app must ensure 100% accuracy in transactions to maintain user trust.

5. **Team Collaboration**:
   - Coordinating work among developers, designers, testers, and stakeholders can be difficult.
   - Example: A global team working on a project may face communication barriers due to time zones.

2. Differentiate Between System Engineering and Software Engineering.

Aspect	System Engineering	Software Engineering
Scope	Focuses on the entire system (hardware, software, processes).	Focuses only on the software component of the system.
Objective	Ensures that all system components work together seamlessly.	Ensures that the software is functional, reliable, and maintainable.
Example	Designing an autonomous car (includes sensors, software, mechanical parts).	Developing the software for the car's navigation system.
Interdisciplinary	Involves multiple disciplines (electrical, mechanical, software).	Primarily involves software development and programming.

Unit 2: Software Development Models

1. Explain the Waterfall Model with Its Advantages and Disadvantages.

Waterfall Model:
The Waterfall model is a linear and sequential approach to software development. It consists of distinct phases:

1. **Requirements**: Gather and document all requirements.
2. **Design**: Create system and software designs.
3. **Implementation**: Write code based on the design.
4. **Testing**: Test the software for defects.
5. **Maintenance**: Fix issues and update the software.

Advantages:

- Simple and easy to understand.
- Works well for small projects with well-defined requirements.
- Clear milestones and deliverables.

Disadvantages:

- Inflexible to changes in requirements.
- Testing is done late in the process, making it costly to fix defects.
- Not suitable for large or complex projects.

Example:
A project to develop a basic calculator app could use the Waterfall model because the requirements are straightforward and unlikely to change.

2. Compare Agile Methods with the Spiral Model.

Aspect	Agile Methods	Spiral Model
Approach	Iterative and incremental.	Combines iterative development with risk analysis.
Flexibility	Highly flexible to changing requirements.	Flexible but focuses on risk management.
Risk Management	Risks are addressed during iterations.	Explicitly addresses risks in each cycle.
Example	Developing a mobile app with frequent updates.	Building a complex system like a satellite.

Unit 3: Requirement Analysis

1. What Are Functional and Non-Functional Requirements? Provide Examples.

Functional Requirements:

- Define what the system should do.
- Example:
  - A login system must allow users to authenticate using a username and password.
  - An e-commerce app must allow users to add items to a cart.

Non-Functional Requirements:

- Define how the system should perform.
- Example:
  - The system must handle 1,000 concurrent users.
  - The app must load in under 2 seconds.

2. Describe the Use-Case Modeling Technique for Requirement Elicitation.

Use-Case Modeling:

- A technique to capture functional requirements by describing interactions between users (actors) and the system.
- Components:

    - **Actors**: Users or external systems interacting with the system.
    - **Use Cases**: Specific functionalities (e.g., "Withdraw Cash").
    - **Relationships**: Associations between actors and use cases.

Example:
For an ATM system:

- Actors: Customer, Bank.
- Use Cases: Withdraw Cash, Check Balance, Transfer Funds.

Unit 4: Software Design

1. Explain the Concepts of Cohesion and Coupling in Software Design.

Cohesion:

- Measures how closely related the functionalities within a module are.
- **High Cohesion**: A module performs a single task (e.g., a module that only calculates taxes).
- **Low Cohesion**: A module performs multiple unrelated tasks (e.g., a module that calculates taxes, prints reports, and sends emails).

Coupling:

- Measures the interdependence between modules.
- **Low Coupling**: Modules are independent (e.g., a tax calculation module does not depend on a report generation module).
- **High Coupling**: Modules are tightly interconnected (e.g., a tax calculation module directly accesses the database).

2. Discuss the Client-Server Architectural Model with a Diagram.

Client-Server Model:

- Divides the system into two components:
    - **Client**: Requests services (e.g., a web browser).
    - **Server**: Provides services (e.g., a web server).

Example:

- A web application where the client (browser) sends requests to the server (web server), which processes the request and returns the response.

Unit 5: Coding

1. What Are the Best Practices for Writing Clean and Maintainable Code?

1. **Use Meaningful Names**:
    - Example: Use `calculateTotal()` instead of `calc()`.
2. **Follow DRY Principle**:
    - Avoid code duplication by reusing functions.
3. **Write Modular Code**:
    - Break code into small, reusable modules.
4. **Add Comments**:
    - Explain complex logic for future developers.
5. **Test Code**:
    - Write unit tests to ensure functionality.

Unit 6: Testing & QA

1. Differentiate Between Black-Box and White-Box Testing.

Aspect	Black-Box Testing	White-Box Testing
Focus	Tests functionality without knowing internal code.	Tests internal logic and structure of the code.
Example	Testing a login feature by entering valid/invalid credentials.	Testing all branches in a login function.

2. Explain the Levels of Testing (Unit, Integration, System).

1. **Unit Testing**:
    - Tests individual components (e.g., a function to calculate tax).
2. **Integration Testing**:
    - Tests interactions between modules (e.g., testing the integration of a payment gateway with an e-commerce app).
3. **System Testing**:
    - Tests the entire system (e.g., testing the complete e-commerce app).

Unit 7: Maintenance

1. What Are the Types of Software Maintenance? Explain with Examples.

1. **Corrective Maintenance**:
    - Fixing bugs or errors.
    - Example: Fixing a login bug in an app.
2. **Adaptive Maintenance**:
    - Adapting software to new environments.
    - Example: Updating an app to support a new OS version.
3. **Perfective Maintenance**:
    - Adding new features or improving performance.
    - Example: Adding a dark mode feature to an app.

Unit 1: Introduction

Questions:

1. What are the key characteristics of good software?
2. Explain the importance of professional ethics in software engineering.

Answers:

1. **Characteristics of Good Software**:
    - **Correctness**: The software should perform its intended functions accurately.
    - **Maintainability**: It should be easy to modify and update.
    - **Usability**: The software should be user-friendly.
    - **Reliability**: It should perform consistently under specified conditions.
    - **Efficiency**: It should use system resources optimally.
2. **Professional Ethics in Software Engineering**:
    - Software engineers must adhere to ethical standards to ensure trust and integrity.
    - Examples:
        - **Confidentiality**: Protecting user data.
        - **Honesty**: Not misrepresenting the capabilities of software.
        - **Sustainability**: Developing software that minimizes environmental impact.

Unit 2: Software Development Models

Questions:

1. What is the Rational Unified Process (RUP)? Explain its phases.
2. Compare Component-Based Software Engineering (CBSE) with Agile methods.

Answers:

1. **Rational Unified Process (RUP)**:
    - A framework for iterative software development.
    - **Phases**:
        - **Inception**: Define project scope and objectives.
        - **Elaboration**: Develop a detailed plan and architecture.
        - **Construction**: Build the software.
        - **Transition**: Deploy the software to users.
2. **CBSE vs. Agile**:
    - **CBSE**: Focuses on reusing pre-built components to speed up development.
    - **Agile**: Focuses on iterative development and customer collaboration.
    - Example: CBSE is used in building enterprise systems, while Agile is used in startups.

Unit 3: Requirement Analysis

Questions:

1. What is the difference between user requirements and system requirements?

2. Explain the role of ethnography in requirement elicitation.

Answers:

1. **User vs. System Requirements**:
    - **User Requirements**: High-level statements of what users need (e.g., "The system should be easy to use").
    - **System Requirements**: Detailed specifications of how the system will meet user needs (e.g., "The system must support 1,000 concurrent users").
2. **Ethnography in Requirement Elicitation**:
    - Ethnography involves observing users in their natural environment to understand their needs.
    - Example: Observing how nurses use a hospital management system to identify pain points.

Unit 4: Software Design

Questions:

1. What is the importance of information hiding in software design?
2. Explain the layered architectural model with an example.

Answers:

1. **Information Hiding**:

    - It ensures that modules hide their internal details, exposing only necessary interfaces.
    - Example: A database module hides its query logic, exposing only a `getData()` function.
2. **Layered Architectural Model**:
    - Divides the system into layers, each with a specific responsibility.
    - Example:
        - **Presentation Layer**: Handles user interface.
        - **Business Logic Layer**: Processes data.
        - **Data Access Layer**: Manages database interactions.

Unit 5: Coding

Questions:

1. Why is code readability important in software development?
2. What are the key principles of the DRY (Don't Repeat Yourself) principle?

Answers:

1. **Code Readability**:
    - Readable code is easier to understand, maintain, and debug.
    - Example: Using meaningful variable names like `totalSalary` instead of `ts`.
2. **DRY Principle**:
    - Avoid duplicating code by reusing functions or modules.

- Example: Instead of writing the same validation logic in multiple places, create a reusable `validateInput()` function.

Unit 6: Testing & QA

Questions:

1. What is regression testing, and why is it important?
2. Explain the concept of test-driven development (TDD).

Answers:

1. **Regression Testing**:
   - Ensures that new changes do not break existing functionality.
   - Example: After adding a new feature to an app, run tests to ensure old features still work.
2. **Test-Driven Development (TDD)**:
   - A development approach where tests are written before the code.
   - Steps:
     - Write a failing test.
     - Write code to pass the test.
     - Refactor the code.

Unit 7: Maintenance

Questions:

1. What is re-engineering in software maintenance?
2. Explain the importance of configuration management in software projects.

Answers:

1. **Re-Engineering**:
    - Restructuring existing software to improve its quality or performance.
    - Example: Migrating a legacy system to a modern platform.
2. **Configuration Management**:
    - Tracks and controls changes to software artifacts (e.g., code, documents).
    - Example: Using Git for version control to manage code changes.

Unit 8: Managing Software Projects

Questions:

1. What are the key activities involved in project planning?
2. Explain the importance of risk management in software projects.

Answers:

1. **Project Planning Activities**:
    - **Scope Definition**: Define project goals and deliverables.
    - **Resource Allocation**: Assign team members and tools.
    - **Scheduling**: Create timelines for tasks.
    - **Budgeting**: Estimate costs and allocate funds.
2. **Risk Management**:
    - Identifies potential risks and plans mitigation strategies.
    - Example: If a key team member leaves, have a backup plan to avoid delays.
