Unit 1: Software Engineering Fundamentals

Short Answer Questions:

Q. Define Software. How are its characteristics different from hardware? (Nov22)

Ans. Software is a set of instructions for a computer to perform tasks, while hardware refers to the physical components of a system; unlike hardware, software is intangible and does not wear out.

Q. How do linear and iterative process models differ? (Nov22)

Ans. Linear models follow a sequential path (e.g., Waterfall), whereas iterative models (e.g., Spiral) allow revisiting and refining phases through repeated cycles.

Q. Write a short note on need of software engineering. (Nov23)

Ans. Software engineering ensures the systematic development, operation, and maintenance of software for higher quality and lower risk.

Q. Describe the concept of prescriptive process models. (Nov23)

Ans. These are structured approaches like Waterfall, Spiral, and V-Model, designed to guide software development in a controlled, step-by-step manner.

Q. What are the key challenges in software development? (Nov24)

Ans. Key challenges include managing changing requirements, ensuring quality, meeting deadlines, and maintaining cost efficiency.

Q. What is structured design methodology? (Nov22)

Ans. It is a technique that uses tools like data flow diagrams (DFDs) to logically organize and break down software into manageable components.

Long Answer Questions

Q. "Specialized process models tend to be applied when a specialized or narrowly defined software engineering approach is chosen." Comment. Discuss the component based development model in detail. Provide three examples of software projects that would be amenable to the component based process model. (Nov22)

Ans. Specialized process models are tailored to address specific project needs, domain requirements, or technological constraints. Unlike generic models, these are chosen when conventional methods do not effectively align with unique project demands such as rapid development, high reliability, or modular construction. One such specialized model is the **Component-Based Development (CBD) Model**.

The CBD model emphasizes the design and construction of computer-based systems using reusable software components. These components are pre-developed, independently deployable units that encapsulate functionality and can be assembled to form a complete system. The CBD process involves component qualification, adaptation, composition, and updating. It enhances productivity, reduces time-to-market, and lowers development costs by reusing existing software modules.

CBD is especially useful in situations requiring modularity and ease of maintenance. It supports scalability and software reuse, which is ideal for large-scale and distributed applications. The model works well with middleware and service-oriented architectures (SOA).

Examples of software projects suitable for CBD:

1. **E-commerce platforms** – where cart, payment, and inventory modules are developed as reusable components.
2. **Hospital management systems** – with patient records, billing, and scheduling modules.
3. **Customer Relationship Management (CRM) software** – integrating modules like contact management, sales tracking, and customer service.

Thus, CBD fosters efficiency and maintainability in component-centric development.

Q. Explain the phases of the unified process in detail. Differentiate between validation testing and system testing. (Nov22)

Ans. The **Unified Process (UP)** is an iterative and incremental software development framework that organizes the project lifecycle into four distinct **phases**, each consisting of multiple iterations:

1. **Inception Phase**: This phase focuses on understanding the project's scope, business case, and feasibility. Key deliverables include initial use-case models, project plans, and risk assessments. The goal is to define the problem and assess whether the project is worth pursuing.
2. **Elaboration Phase**: The system's architecture and design are refined. High-risk elements are tackled first, and more detailed requirements are gathered. Key deliverables include an executable architectural baseline and refined risk assessments.
3. **Construction Phase**: The software is built and integrated. Major coding takes place, including implementation of features, components, and subsystems. Iterations result in fully functional versions of the product.
4. **Transition Phase**: The software is released to the end-users. It includes beta testing, bug fixing, and final adjustments based on user feedback. User training and documentation are also provided.

Difference Between Validation Testing and System Testing:

- **Validation Testing** ensures the software meets user needs and requirements (i.e., "Are we building the right product?").
- **System Testing** verifies the complete and integrated software system to ensure it complies with specified requirements (i.e., "Are we building the product right?").

Both are essential for delivering reliable, user-accepted software.

Unit 2: Requirements Analysis
Short Answer Questions

Q. Which of the development process models would you employ for developing a web-based system for a new business where requirements are changing fast and where an in-house development team is available for all aspects of the project? Justify your answer. (Nov22)

Ans. **Development Model for Evolving Web Systems:** For rapidly changing requirements in a web-based system, the **Agile model** is ideal due to its iterative development, frequent feedback, and flexibility to adapt changes quickly.

Q. What is the primary role of a system analyst? (Nov23),(Nov24)

Ans. **Role of System Analyst:** A system analyst acts as a bridge between users and developers, responsible for gathering requirements, analyzing problems, and designing appropriate software solutions.

Q. What is Decision tree? (Nov23)

Ans. **Decision Tree:** A decision tree is a graphical representation of possible solutions to a decision based on various conditions, useful for analyzing and predicting outcomes.

Q. Discuss Decision Table. (Nov24)

Ans. **Decision Table:** A decision table is a tabular method for representing and analyzing complex business rules and logic by listing conditions and their corresponding actions.

Q. What are formal requirements? (Nov23)

Ans. **Formal Requirements:** Formal requirements are precise, mathematically-based specifications that define system behavior to ensure accuracy, completeness, and consistency in software design.

Long Answer Questions

Q. What is Software Requirements Specification (SRS)? What are the properties of good SRS documentation? (Nov23), (Nov24)

Ans. A **Software Requirements Specification (SRS)** is a formal document that comprehensively describes the functional and non-functional requirements of a software system. It serves as a contract between the client and the development team, detailing what the software should do, how it should behave, and any constraints it must operate under. The SRS ensures that all stakeholders have a clear and mutual understanding of the system's expectations before development begins.

Properties of a Good SRS:

1. **Correctness**: Every requirement stated must be accurate and reflect the client's needs.
2. **Unambiguity**: Requirements should have only one possible interpretation to avoid confusion.
3. **Completeness**: The document should include all necessary requirements, including use cases, diagrams, and interfaces.
4. **Consistency**: There should be no conflicting requirements within the document.
5. **Verifiability**: Each requirement must be testable through inspection, analysis, or testing.
6. **Modifiability**: The structure should allow easy updates without affecting unrelated requirements.
7. **Traceability**: Each requirement should be easily traceable to its origin and throughout the project lifecycle.

A well-structured SRS improves communication, reduces development errors, and serves as a reference throughout the software development process.

Q. What are the basic issues that an SRS must address? Differentiate between functional and non-functional requirements. (Nov24),(Nov22)

Ans. A **Software Requirements Specification (SRS)** must address several basic issues to effectively guide the development process and meet stakeholder expectations. These include:

1. **Functionality**: What the system should do—its features, services, and tasks.
2. **Interfaces**: Interactions with other software, hardware, users, and systems.
3. **Performance**: Speed, responsiveness, scalability, and resource usage.

4. **Reliability and Availability**: How dependable the system must be.
5. **Security**: Measures to protect data and functionality.
6. **Compliance**: Adherence to legal, regulatory, and industry standards.
7. **Constraints**: Limitations like platform, hardware, or programming language.
8. **Assumptions and Dependencies**: External conditions or systems the software relies on.

Functional vs. Non-Functional Requirements:

- **Functional Requirements** describe **what** the system must do. They define specific behaviors or functions, such as:
  - "The system must generate monthly sales reports."
  - "Users must be able to reset their passwords."
- **Non-Functional Requirements** specify **how** the system performs its functions. These include:
  - **Performance** (e.g., response time < 2 seconds),
  - **Usability**, **Security**, **Maintainability**, and **Scalability**.

While functional requirements define features, non-functional ones define quality attributes, and both are critical for a complete and useful SRS.

Q. What are the objectives of Requirement Analysis? (Nov24)

Ans. **Requirement Analysis** is a crucial phase in the software development lifecycle that focuses on identifying, understanding, and documenting what a software system must accomplish. The primary **objectives of requirement analysis** include:

1. **Understanding User Needs**: The foremost goal is to capture what users truly need from the system, ensuring their expectations are accurately translated into software functionality.
2. **Defining System Boundaries**: It helps define what will be included in the system and what will not, clarifying system scope and avoiding scope creep during development.
3. **Creating Clear Requirements**: It aims to produce precise, unambiguous, complete, and consistent requirement specifications that developers and stakeholders can agree upon.
4. **Identifying Constraints**: Requirement analysis uncovers limitations such as technical, legal, budgetary, or time-related constraints that might affect the project.
5. **Improving Communication**: It provides a foundation for effective communication between stakeholders, project managers, and developers by documenting expectations clearly.
6. **Prioritizing Requirements**: Not all features are equally important; this process helps in prioritizing what should be developed first based on user needs and business value.

By achieving these objectives, requirement analysis ensures that the final system is useful, cost-effective, and aligned with user expectations, reducing risks of failure and rework later.

Q. Describe some commonly used techniques for software cost estimation. (Nov23), (Nov24)

Ans. **Software cost estimation** is the process of predicting the effort, time, and financial resources required to develop a software system. Accurate cost estimation is critical for project planning and budgeting. Here are some commonly used techniques:

1. **Expert Judgment**: Involves consulting experienced project managers or developers who use their past experience to estimate costs. It's quick but subjective.
2. **Analogous Estimation**: Compares the current project with similar completed projects to derive estimates. It relies on historical data but may lack accuracy if projects differ significantly.
3. **Top-down Estimation**: Starts with an overall estimate for the project and breaks it down into smaller components. It's useful in early stages but may overlook details.
4. **Bottom-up Estimation**: Estimates are made for each small task or module and then aggregated. This method is detailed and accurate but time-consuming.
5. **Parametric Estimation (e.g., COCOMO Model)**: Uses mathematical models based on historical data and project parameters like size (measured in LOC or function points). The COCOMO (Constructive Cost Model) is widely used.
6. **Use Case Point Estimation**: Estimates cost based on the number and complexity of use cases. It's useful in object-oriented systems.

Each technique has its pros and cons, and often a combination is used for reliable estimation.

Unit 3: Software Design

Short Answer Questions:

Q. Why is modularity a desired characteristic in software design? (Nov22)
Ans. Modularity breaks software into independent, manageable units or modules, enhancing readability, maintainability, and debugging efficiency.

Q. What is Coupling? (Nov24)
Ans. Coupling measures the interdependence between software modules; low coupling is preferred for flexible and maintainable systems.

Q. Differentiate between data coupling and control coupling. (Nov22)
Ans. Data coupling occurs when modules communicate by passing only required data, while control coupling happens when control flags or logic are passed, leading to tighter dependency.

Q. Discuss the major advantages of OOD approach over the function-oriented design approach. (Nov22)
Ans. Object-Oriented Design (OOD) promotes code reusability, modularity, and better abstraction, making maintenance and scalability easier than traditional function-based approaches.

Q. Write a short note on ER diagram. (Nov24)
Ans. An Entity-Relationship (ER) diagram visually represents entities, relationships, and attributes in a database system, aiding in database structure design.

Q. Explain object oriented design. (Nov24)
Ans. Object-Oriented Design models a system using classes and objects, encapsulating data and behavior for improved modularity and code reuse.

Long Answer Questions:
Q. What do you mean by cohesion in software design? Discuss the structure of SRS. Discuss various levels of cohesion with suitable examples. (Nov23), (Nov22)
Ans. Cohesion in software design refers to the degree to which the elements within a module belong together and work toward a single, well-defined purpose. High cohesion is desirable as it leads to more understandable, maintainable, and reusable code.
Structure of SRS (Software Requirements Specification):
An SRS typically includes:

- Introduction: Overview, purpose, scope, and definitions.
- Overall Description: Product perspective, features, user needs, and assumptions.
- Specific Requirements: Functional, non-functional, interface, and performance requirements.
- Appendices: Supporting information like diagrams, references, and glossary.
- Index: For easy navigation.

Levels of Cohesion:

1. Coincidental Cohesion: Elements are grouped arbitrarily (e.g., a utility module with unrelated functions).
2. Logical Cohesion: Elements perform similar activities (e.g., input routines based on user action).
3. Temporal Cohesion: Elements executed at the same time (e.g., initialization routines).
4. Procedural Cohesion: Elements follow a specific control sequence (e.g., processing a form).
5. Communicational Cohesion: Elements operate on the same data set (e.g., functions using the same file).
6. Sequential Cohesion: Output from one part is input to another (e.g., pipeline processing).
7. Functional Cohesion: All elements contribute to a single task (e.g., calculating interest).

Functional cohesion is the most desirable, as it ensures clarity and efficiency in software modules.

Q. What are the characteristics of a good software design? Define the three software design complexity measures: structural complexity, data complexity, and system complexity with the help of examples. (Nov22)
Ans. A good software design should possess the following key characteristics:

1. **Correctness** – It must fulfill all specified requirements.
2. **Understandability** – The design should be easy to understand and explain.
3. **Efficiency** – It must use system resources effectively.
4. **Modularity** – The design should be divided into independent components.
5. **Maintainability** – It should allow easy updates and modifications.
6. **Reusability** – Components should be usable in different applications.
7. **Scalability** – The design must adapt to increasing loads or expansions.
8. **Robustness** – It should handle unexpected inputs or failures gracefully.

Software Design Complexity Measures:

1. **Structural Complexity**: Measures how modules interact. A highly interconnected module system (high fan-in/fan-out) increases structural complexity.
   Example: A module calling 10 other modules is more complex than one calling only 2.
2. **Data Complexity**: Relates to how modules handle input/output data. A module using many global variables or data formats is considered more data complex.
   Example: A module processing multiple file types is more data complex than one processing a single file format.
3. **System Complexity**: It is a combination of structural and data complexity, reflecting the overall intricacy of the software system.
   Example: A system with many interrelated modules and diverse data flows has high system complexity.

Keeping complexity low helps in achieving more robust and maintainable software.

Q. What is design process in software engineering? What is the importance of a good design? (Nov23), (Nov24)

Ans. In **Software Engineering**, the **design process** is the step following requirements analysis and precedes implementation. It focuses on **transforming user requirements into a suitable blueprint** that developers can use to build the system. This involves defining system architecture, components, interfaces, data structures, and algorithms.

The **design process** typically includes:

1. **Architectural Design** – Lays out the overall structure and identifies major modules and their interactions.
2. **High-Level Design (HLD)** – Describes modules in more detail including their functions and data flow.
3. **Low-Level Design (LLD)** – Details the logic, algorithms, and data structures within each module.

Importance of a Good Design:

1. **Foundation for Development** – A well-structured design provides a clear roadmap for developers.
2. **Reduces Complexity** – By breaking down the system into manageable modules, it becomes easier to understand and build.
3. **Improves Maintainability** – Clean and modular design simplifies debugging, testing, and future updates.
4. **Enhances Reusability** – Well-designed components can be reused in other projects.
5. **Facilitates Communication** – Design documents help in clear communication among team members and stakeholders.
6. **Early Error Detection** – Design review can catch issues before coding begins, saving time and cost.

In summary, a good software design ensures system quality, reliability, and long-term success.

Q. Explain the key principles and advantages of Object-Oriented Design in software development. (Nov23)

Ans. **Object-Oriented Design (OOD)** is a software development approach that focuses on modeling a system using real-world entities called "objects." Each object represents a specific entity that has attributes (data) and behaviors (methods). OOD is based on several key principles:

Key Principles:

1. **Encapsulation** – Bundling data and methods that operate on the data within a single unit (object), protecting internal states.
2. **Abstraction** – Hiding complex implementation details and showing only essential features to the user.
3. **Inheritance** – Allowing one class to inherit attributes and methods from another, promoting code reuse.
4. **Polymorphism** – Enabling objects to be treated as instances of their parent class, allowing for method overriding and overloading.
5. **Modularity** – Breaking the system into smaller, manageable, and independent units (classes or objects).

Advantages:

- **Code Reusability** – Inheritance allows the reuse of existing code, reducing redundancy.

- **Maintainability** – Modular structure makes it easier to test, update, and debug code.
- **Scalability** – Systems can be easily extended with new objects or features.
- **Real-World Modeling** – Objects represent real-world entities, making design intuitive and easier to understand.
- **Improved Collaboration** – Developers can work on different objects/classes independently.

In summary, Object-Oriented Design enhances software quality, flexibility, and clarity, making it a popular choice for complex and evolving systems.

Q. What is Function Oriented Design, and how does it contribute to the software development process? (Nov24)

Ans. **Function Oriented Design (FOD)** is a traditional software design methodology where the system is decomposed into a set of interacting functions or modules. It emphasizes processes and operations over data. The main goal is to transform inputs into outputs through well-defined functions, focusing on "what the system does" rather than "what the system contains."

Key Characteristics of FOD:

- **Top-down approach:** The design starts from a general overview and breaks down into smaller, more manageable modules.
- **Data Flow Diagrams (DFDs):** Used to represent the flow of data and the processes that transform the data.
- **Modularity:** The system is broken into multiple modules with clearly defined functionality.
- **High cohesion and low coupling:** Each function performs a single task and interacts minimally with other functions.

Contribution to Software Development:

- FOD helps in **systematic analysis and documentation**, which improves understanding and communication among development teams.
- It **simplifies debugging and maintenance** by isolating functionality into independent modules.
- It is **suitable for small and medium-sized systems** where data structures are relatively simple and functionality is the focus.

However, FOD has limitations when it comes to handling complex data relationships, which is why many modern systems use **Object-Oriented Design (OOD)** instead. Still, FOD provides a solid foundation for procedural programming and process-based applications.

Unit 4: Testing and Maintenance

Short Answer Questions:

Q. Discuss the need of software maintenance. (Nov22)
ANS. Software maintenance is essential to correct errors, improve performance, and adapt the software to changing user requirements or environments.

Q. Write a short note on unit testing. (Nov23), (Nov24)
Ans. Unit testing involves testing individual components or functions of a software program to ensure they work as intended in isolation.

Long Answer Questions:

Q. Write a detailed note on software metrics used to estimate testing effort. What is the software maturity index and what is it used for? (Nov23), (Nov24), (Nov22), (Nov22), (Nov23)
Ans. **Software Metrics for Estimating Testing Effort:**
Software metrics are essential tools used to estimate the time, cost, and resources required for testing in software projects. Key metrics used for estimating testing effort include:

1. **Test Case Metrics:** Measure the number of test cases planned, executed, passed, or failed. Helps estimate total testing workload.
2. **Defect Density:** Number of defects per KLOC (thousand lines of code). Indicates code quality and the amount of testing required.
3. **Requirements Coverage:** Measures how many requirements are covered by test cases. Higher coverage requires more effort but ensures quality.
4. **Code Complexity (Cyclomatic Complexity):** Higher complexity means more paths to test, thus increasing testing effort.
5. **Test Point Analysis (TPA):** Quantifies test size based on system functionality, complexity, and quality requirements.

These metrics guide resource allocation and timeline planning for the testing phase.

Software Maturity Index (SMI):
The **Software Maturity Index (SMI)** is a metric that reflects the stability and maturity of a software product. It is calculated using:
SMI = [Mt - (Fa + Fc + Fd)] / Mt
Where:

- **Mt** = Total modules in the current release
- **Fa** = Modules added
- **Fc** = Modules changed
- **Fd** = Modules deleted

An SMI value closer to 1 indicates a more stable, mature software system. It is used to monitor project progress and readiness for release.

Q. Describe the main goals of System Testing and explain how it works? (Nov23)
Ans. **System Testing: Goals and Functionality**
System testing is a crucial phase of the software testing life cycle (STLC) where the complete integrated system is tested to evaluate its compliance with the specified requirements. It is a **black-box testing** method conducted after integration testing and before acceptance testing.
Main Goals of System Testing:

1. **Validation of Functional Requirements:** Ensure that all user requirements and business functions are working as intended.
2. **End-to-End Testing:** Verify the behavior of the entire system under real-world scenarios.
3. **Detection of Defects:** Identify bugs or inconsistencies in the integrated system.
4. **Performance and Security Testing:** Test how the system performs under load and handles data securely.
5. **System Compliance:** Check compatibility with external interfaces and adherence to regulatory standards.

How System Testing Works:

- It begins once all modules are integrated and a complete build is available.
- Test cases are derived from SRS (Software Requirement Specification).
- Various types of system tests are conducted, including:
    - Functional testing
    - Usability testing
    - Load and stress testing
    - Recovery testing
- Tools may be used to automate test execution.
- Results are documented and any defects are reported to developers for resolution.

System testing ensures that the software is reliable and ready for deployment.

What is Reengineering of Software? How it is used for recreating a design? (Nov23),(Nov22),(Nov24)

Ans. **Reengineering of Software: Definition and Role in Design Recreation**

Software reengineering is the process of analyzing and modifying existing software systems to improve functionality, performance, or maintainability without changing their core functionality. It involves transforming legacy systems into more efficient, modernized systems by refining their design and structure.

Purpose of Software Reengineering:

- To enhance the quality and performance of outdated or poorly structured systems.
- To reduce maintenance costs and improve scalability.
- To adapt software to new technologies or business requirements.

How It Helps in Recreating Design:

1. **Reverse Engineering:** The first step is to analyze the existing software to extract design and requirements information. This helps in understanding the system's architecture, logic, and dependencies.
2. **Design Recovery:** Based on reverse engineering, the system's original design is reconstructed, often using models or documentation that may have been lost over time.
3. **Forward Engineering:** Once the new design is prepared, the software is rebuilt using modern tools and coding standards, incorporating enhancements while retaining core functions.
4. **Refactoring and Restructuring:** Code and database structures are cleaned up, modules are modularized, and redundancy is eliminated.

Reengineering supports future growth and innovation by enabling legacy systems to align with current business and technology trends.

Q. Explain the concept of Software Measurement and its significance in managing software.(Nov23), (Nov24), (Nov24)

Ans. **Software Measurement and Its Significance**

Software measurement is the process of quantitatively assessing various attributes of software products, processes, and resources throughout the development lifecycle. It involves assigning numerical values (metrics) to elements such as code complexity, development effort, defects, performance, and maintainability. These measurements help in understanding, controlling, and improving the software development process.

Key Aspects Measured:

- **Product metrics:** Size (LOC), complexity (Cyclomatic Complexity), and quality (defect density).
- **Process metrics:** Development time, productivity, and efficiency of different phases.
- **Project metrics:** Cost, schedule adherence, and resource utilization.

Significance in Software Management:

1. **Project Planning and Estimation:** Helps in predicting resources, time, and cost required for future projects.
2. **Quality Assurance:** Identifies areas of high risk or poor quality, enabling early corrective actions.
3. **Process Improvement:** Metrics highlight inefficiencies in the development process and help managers adopt better methodologies.
4. **Performance Monitoring:** Tracks team and individual productivity, ensuring accountability and better workload distribution.
5. **Decision-Making:** Empirical data supports objective decisions on technology adoption, outsourcing, or design changes.

In essence, software measurement enables better control over software projects, ensuring they are delivered on time, within budget, and at the desired quality.