SOFTWARE ENGINEERING - 10MARKS

2019

1. Why is software testing required? Differentiate between White box testing and Black box testing.

(4 + 6 marks)

Why is software testing required? (4 marks)

Software testing is an essential part of the software development lifecycle that ensures the software is working as expected and meets the user requirements. The primary purpose of testing is to identify and fix bugs or issues that may affect the functionality, security, and performance of the software.

Key reasons why software testing is required:

1. **Ensures Quality:** Testing helps verify that the software performs as intended and meets the quality standards.
2. **Identifies Defects:** It helps in identifying bugs, issues, and flaws in the system before it is deployed to users.
3. **Improves Reliability:** Thorough testing ensures that the software is stable and reliable, minimizing the risk of failures in production.
4. **Customer Satisfaction:** Well-tested software provides confidence to users, ensuring their needs and expectations are met.
5. **Cost-Efficiency:** Identifying and fixing issues early in the development cycle is more cost-effective than dealing with problems after deployment.

Differentiate between White box testing and Black box testing. (6 marks)

White Box Testing and **Black Box Testing** are two fundamental testing techniques used to evaluate software.

Aspect	White Box Testing	Black Box Testing
Definition	White box testing focuses on the internal workings and structure of the software. It tests individual functions, code paths, and logic based on the knowledge of the program's source code.	Black box testing evaluates the software's functionality from the user's perspective without knowledge of the internal code or logic. It focuses on verifying whether the software meets the specified requirements.
Test Basis	Based on the internal design and implementation of the software.	Based on the external specifications and functional requirements of the software.
Testers	Testers need knowledge of the code and system design (developer-oriented).	Testers do not need knowledge of the internal workings (user-oriented).
Type of Testing	Includes testing code, functions, conditions, loops, and data flow.	Includes functional testing, system testing, and acceptance testing.

Aspect	White Box Testing	Black Box Testing
Advantages	Provides a deep insight into the program's internal workings. It helps in detecting hidden errors and optimizing the code.	Helps in evaluating the software's usability and correctness from the user's perspective. It doesn't require knowledge of the source code.
Example	Unit testing, integration testing, and security testing are examples of white box testing.	Functional testing, regression testing, and user acceptance testing (UAT) are examples of black box testing.

Summary:

- **White box testing** is code-focused, testing internal structures and logic, while **Black box testing** is behavior-focused, testing the system's outputs based on inputs.

---

2. What is software design? Explain objectives and principles of software design.

(3 + 7 marks)

What is Software Design? (3 marks)

Software design is the process of defining the architecture, components, interfaces, and data for a software system to satisfy specified requirements. It is a blueprint for the construction of the software. Software design takes the requirements gathered and translates them into a detailed solution for developers to implement.

- **Levels of Software Design:**
    - **High-level design (Architectural Design):** Focuses on the overall structure of the software system and the interaction between its components.
    - **Low-level design (Detailed Design):** Focuses on the design of individual components, classes, and methods.

Objectives and Principles of Software Design (7 marks)

Objectives of Software Design:

1. **Simplicity:** The design should be as simple as possible while fulfilling all requirements. A complex design leads to higher maintenance costs and errors.
2. **Modularity:** The system should be divided into smaller, independent modules. Each module should have a single responsibility (Single Responsibility Principle).
3. **Scalability:** The design should accommodate future growth, such as increasing load, adding new features, or supporting larger data sets.
4. **Maintainability:** The design should allow for easy updates, bug fixes, and adaptations to meet new requirements.
5. **Efficiency:** The system should make optimal use of available resources, such as memory and processing power, to ensure good performance.

6. **Reusability:** The design should allow for components and modules to be reused across different parts of the system or in future projects.
7. **Testability:** The system should be designed in a way that allows easy testing of individual modules and components.

Principles of Software Design:

1. **Separation of Concerns:** Divide the system into distinct features that overlap in functionality as little as possible. This makes it easier to maintain and test the system.
2. **Encapsulation:** Hide the internal details of a module and expose only what is necessary for other modules to interact with. This reduces complexity and potential errors.
3. **Abstraction:** Provide a simplified interface for interacting with the software while hiding the complex implementation details.
4. **Modularity:** Break the system into smaller, manageable modules. Each module should be self-contained and interact with other modules only through well-defined interfaces.
5. **Reusability:** Design components in such a way that they can be reused across different projects, reducing redundancy and improving consistency.
6. **Flexibility and Extensibility:** The design should be flexible enough to accommodate changes and extensions without major rework.
7. **Consistency:** Use consistent naming conventions, design patterns, and architecture throughout the system to improve readability and reduce complexity.

Summary:
The main goal of software design is to create a system that is **easy to develop**, **maintain**, and **scale**, while following key principles to ensure reliability, efficiency, and ease of understanding.

2021

1. Define testing. What is testability of software? What is unit testing? Define verification and validation.

(2 + 2 + 3 + 3 marks)

Define Testing (2 marks)

Software testing is the process of evaluating a software application to identify any defects or issues. It involves executing the software to check if it behaves as expected and meets the required specifications. Testing helps ensure the software's quality, reliability, and functionality by detecting bugs early in the development process.

What is Testability of Software? (2 marks)

Testability refers to how easily a software system can be tested. A system is considered highly testable if it allows for efficient identification of defects and is easy to verify against the specified requirements. Testability depends on factors such as the software's design, modularity, documentation, and the availability of clear interfaces for testing.

- **Factors Affecting Testability:**

1. Clear and well-defined requirements.
2. Modular design.
3. Availability of automated testing tools.

What is Unit Testing? (3 marks)

Unit testing is the process of testing individual components or units of a software application, such as functions, methods, or classes, to ensure they work as expected. It is typically done during the development phase by the developers.

- **Goal:** The goal of unit testing is to catch errors in the smallest testable parts of the application.
- **Tools:** Unit testing can be automated using frameworks like **JUnit** for Java, **NUnit** for .NET, and **pytest** for Python.
- **Importance:** Unit tests help ensure that each part of the software works in isolation and that defects are detected early, improving overall software quality.

Define Verification and Validation. (3 marks)

1. **Verification**
   Verification is the process of evaluating whether the software meets the specified requirements and design specifications. It answers the question, **"Are we building the product right?"**
   It ensures that the product is being developed according to the correct specifications and standards.
   - **Example:** Reviewing design documents, code reviews, and walkthroughs.
2. **Validation**
   Validation is the process of evaluating the software in its final form to ensure it meets the user's needs and requirements. It answers the question, **"Are we building the right product?"**
   Validation confirms that the software does what the users expect and delivers the desired outcomes.
   - **Example:** User acceptance testing (UAT) and beta testing.

---

2. Define product and process. What are the main Software Quality Assurance activities?

(5 + 5 marks)

Define Product and Process (5 marks)

1. **Product**
   A **product** in the context of software refers to the **end result** of the software development process. It is the software application or system delivered to the users after being designed, developed, tested, and deployed.
   - **Example:** A mobile app, website, or an enterprise software system.

The product is the outcome of the process and is evaluated for functionality, usability, performance, and security.

2. **Process**
   A **process** in software refers to the series of **activities** and **steps** followed during the software

development lifecycle to create a product. This includes planning, requirements gathering, design, development, testing, deployment, and maintenance.

    - **Example:** The Agile or Waterfall process used to develop software.

The process focuses on how the product is built, and its efficiency impacts the quality of the final product.

What are the main Software Quality Assurance activities? (5 marks)

Software Quality Assurance (SQA) is a set of activities aimed at ensuring the quality of software throughout its development lifecycle. The key activities involved in SQA include:

1. **Requirements Review:**
   Ensures that the software requirements are clear, complete, and testable. It is the first step in ensuring the software's quality by verifying if the requirements meet the needs of the stakeholders.
2. **Design Review:**
   Involves evaluating the software design to ensure that it meets the functional and non-functional requirements. Design reviews ensure that the system architecture is sound and scalable.
3. **Code Reviews and Inspections:**
   This activity involves peer reviews of the source code to identify bugs, vulnerabilities, or deviations from coding standards early in the development cycle.
4. **Testing Activities:**
   Involves various levels of testing such as unit testing, integration testing, system testing, and user acceptance testing. SQA ensures that the tests are comprehensive and that defects are reported and resolved.
5. **Process Audits and Continuous Improvement:**
   SQA ensures that the development process follows standards and best practices, conducting audits and assessments to identify areas for improvement. This leads to better efficiency, fewer defects, and higher-quality software.

Summary:
Software quality assurance activities are critical in ensuring that the software is developed in a systematic, disciplined, and quality-focused manner. These activities help identify potential defects early, improve the process, and ensure the product meets user expectations.

2022

1. Explain Different Testing Methods *(10 Marks)*

Software testing methods are used to ensure that a software product functions correctly, meets requirements, and is free of defects. There are several types of testing methods, mainly categorized as:

---

1. Static Testing

- **Definition:** Testing without executing the code.
- **Purpose:** To find errors in design, documentation, or code early in the development process.
- **Examples:** Reviews, walkthroughs, inspections, static analysis tools.

---

2. Dynamic Testing

- **Definition:** Testing by executing the software.
- **Purpose:** To check software behavior during runtime.
- **Types:**
    - **Unit Testing:** Tests individual functions or modules.
    - **Integration Testing:** Checks interactions between modules.
    - **System Testing:** Verifies the complete system against requirements.
    - **Acceptance Testing:** Validates software with user requirements before delivery.

---

3. Black Box Testing

- **Focus:** Tests the software's functionality without looking at internal code.
- **Used for:** Functional testing, system testing, user acceptance testing.
- **Techniques:** Equivalence partitioning, boundary value analysis.

---

4. White Box Testing

- **Focus:** Tests the internal logic and structure of the code.
- **Used for:** Unit testing, code coverage, path testing.
- **Requires:** Knowledge of source code.

---

5. Grey Box Testing

- **Focus:** Combination of black box and white box testing.
- **Tester:** Has partial knowledge of internal workings.
- **Benefit:** Balances structure-based and behavior-based testing.

Conclusion:

Each method targets different aspects of quality—**static testing** finds early defects, **dynamic testing** checks behavior, while **black/white/grey box testing** cover functionality, logic, and integration. Using multiple testing methods leads to higher software reliability and quality.

---

2. What are Software Metrics? How can you measure the quality of software? Discuss different software metrics for software measurement.

(2 + 4 + 4 marks)

What are Software Metrics? (2 marks)

Software metrics are quantitative measures used to assess various aspects of software development, including code quality, performance, complexity, and productivity. They provide valuable insights into the development process and help in decision-making, risk assessment, and project management.

- **Purpose:** Software metrics help in monitoring the progress of development, predicting future performance, and improving the quality of the software by identifying areas for improvement.

How can you measure the quality of software? (4 marks)

Measuring software quality involves evaluating various attributes that define how well the software meets the needs of users and stakeholders. Several factors contribute to software quality, including functionality, performance, security, maintainability, and user satisfaction.

To measure software quality, we can use various **quality attributes** and **metrics**, such as:

1. **Functionality:**
   Measures if the software performs the required tasks and meets user requirements.
2. **Performance:**
   Evaluates how efficiently the software operates, including response times, throughput, and resource consumption.
3. **Reliability:**
   Measures the software's ability to perform without failure over time.
4. **Maintainability:**
   Assesses how easy it is to modify, update, and fix the software after it has been deployed.
5. **Security:**
   Ensures the software is protected against potential security threats, such as unauthorized access and data breaches.
6. **Usability:**
   Evaluates the software's ease of use, user interface design, and user experience.

By using metrics like **defect density**, **code coverage**, and **response time**, the overall quality of the software can be effectively measured.

Discuss Different Software Metrics for Software Measurement. (4 marks)

Software metrics are categorized into different types based on their focus areas. Below are the key metrics used to measure various aspects of the software:

1. **Size Metrics:**
   - **Lines of Code (LOC):** Measures the size of the codebase. While it's easy to calculate, it doesn't always reflect quality.

- **Function Points:** Measures software size based on functionality. It considers inputs, outputs, inquiries, and internal data processing. It’s a more reliable measure than LOC for estimating effort and complexity.

2. **Complexity Metrics:**
    - **Cyclomatic Complexity:** Measures the complexity of a program by counting the number of linearly independent paths through the program. Higher complexity means harder-to-maintain code.
    - **Halstead Complexity Metrics:** Focuses on the volume, difficulty, and effort required to understand a program, based on the number of operators and operands.
3. **Quality Metrics:**
    - **Defect Density:** Measures the number of defects per unit of code (e.g., defects per 1,000 lines of code). It helps assess software quality and reliability.
    - **Code Coverage:** Measures the percentage of the codebase tested by automated test cases. High code coverage indicates better testing of the software.
4. **Maintainability Metrics:**
    - **Maintainability Index:** Combines various metrics like cyclomatic complexity and lines of code to provide an overall measure of maintainability. A higher value indicates easier maintainability.
    - **Refactoring Frequency:** Measures how often the code is refactored to improve structure or efficiency, impacting long-term maintainability.
5. **Performance Metrics:**
    - **Response Time:** Measures the time taken by the system to respond to a user input or request.
    - **Throughput:** The amount of work the software can process in a given time, typically used in performance testing.

2023

1. Define testing. What is testability of software? What is unit testing? Define white-box and black-box testing.

Define Testing

Testing is the process of executing a software application to find errors and ensure it works as expected. It helps verify that the software meets user requirements and performs correctly under different conditions.

What is Testability of Software?

Testability refers to how easily a software system can be tested. High testability means it's easy to design test cases, execute them, and observe the results. Factors like good documentation, modular design, and simple logic improve testability.

What is Unit Testing?

Unit testing involves testing individual components or functions of a program in isolation. It is done by developers to ensure each small part of the software works correctly before combining it with others.

- **Example:** Testing a function that calculates total price from quantity and unit cost.

Define White-box Testing

White-box testing checks the internal logic and structure of the code. Testers need to understand the source code and test paths, loops, and conditions.

- **Example:** Checking if all branches in a decision-making block are covered.

Define Black-box Testing

Black-box testing focuses on software functionality without knowing its internal code. Testers provide inputs and check if outputs match expected results.

- **Example: Testing if a login form works correctly with valid and invalid credentials.**

---

2. Write short notes on:

a. DFD (Data Flow Diagram)

A **Data Flow Diagram (DFD)** is a graphical representation that shows how data moves through a software system. It helps in understanding the system's functionality and how inputs are transformed into outputs. DFDs are used in the early stages of system design and are useful for both technical and non-technical users.

Key Components of a DFD:

1. **Processes:** Represent tasks or operations that transform data. Shown as circles or ovals.
2. **Data Flows:** Arrows showing the movement of data between processes, data stores, and external entities.
3. **Data Stores:** Places where data is held for later use, such as databases or files.
4. **External Entities:** People, systems, or organizations that interact with the system but are outside its boundary (also called sources or sinks).

Levels of DFD:

- **Level 0 (Context Diagram):** Shows the system as a single process and its interactions with external entities.
- **Level 1 and beyond:** Break the system down into sub-processes for more detail.

Uses of DFD:

- Helps in analyzing existing systems.
- Used for planning new systems.
- Makes communication between users and developers easier.

---

b. SRS (Software Requirements Specification)

A **Software Requirements Specification (SRS)** is a document that describes the complete behavior and requirements of a software system. It is prepared after the requirements gathering phase and serves as a reference for both developers and clients.

Main Contents of SRS:

1. **Functional Requirements:** Define what the system should do (e.g., user login, data processing).
2. **Non-Functional Requirements:** Define how the system should perform (e.g., speed, security, reliability).
3. **Interfaces:** Describes user interfaces, hardware, software, or communication interfaces.
4. **Constraints:** Any design limitations, technologies to be used, or legal/compliance issues.
5. **Use Case Scenarios:** Examples showing how users interact with the system.

Importance of SRS:

- Acts as a clear agreement between stakeholders.
- Helps developers understand what to build.
- Serves as a basis for testing and validation.
- Reduces development errors caused by misunderstandings.

2023-24

1. Why should we study software engineering? Why do we use a life cycle model? Write about data flow-oriented design and object-oriented design.

(2 + 3 + 5 marks)

Why should we study software engineering? *(2 marks)*

Studying **software engineering** helps us understand how to develop reliable, efficient, maintainable, and cost-effective software. It provides structured methods, tools, and principles for managing complex software projects, reducing errors, and ensuring that the final product meets user needs.

Why do we use a life cycle model? *(3 marks)*

A **software life cycle model** provides a structured approach to software development. It defines the stages (like planning, designing, coding, and testing) to follow in order to build software systematically.

Reasons for using a life cycle model:

- Ensures better planning and control.
- Helps manage large, complex projects.
- Makes it easier to track progress and identify issues early.
- Provides clear roles and deliverables for each phase.

Data Flow-Oriented Design vs. Object-Oriented Design *(5 marks)*

1. Data Flow-Oriented Design:

- Focuses on how data moves through the system.
- The system is designed using **Data Flow Diagrams (DFDs)**.
- Emphasizes processes and transformations applied to data.
- Good for understanding data processing steps in business applications.

Example: In an ATM system, DFDs show how user input like PIN and amount flows through authentication and transaction processing modules.

2. Object-Oriented Design (OOD):

- Focuses on modeling the system as a group of interacting **objects**.
- Objects have **attributes** (data) and **methods** (functions).
- Uses concepts like **inheritance**, **encapsulation**, and **polymorphism**.
- Promotes code **reusability**, **modularity**, and easier maintenance.

Example: In a school system, classes like Student, Teacher, and Course interact with each other using defined methods.

---

2. Write in brief about the waterfall model. Write the difference between coding and unit testing.

(5 + 5 marks)

Waterfall Model (5 marks)

The **Waterfall model** is a linear and sequential approach to software development. Each phase must be completed before moving to the next, with minimal overlap.

Phases in the Waterfall Model:

1. **Requirement Gathering:** Understand what the software should do.
2. **System Design:** Plan system architecture and components.
3. **Implementation (Coding):** Write the actual program code.
4. **Testing:** Check the software for errors and bugs.
5. **Deployment:** Release the software to users.
6. **Maintenance:** Fix any issues and update the software over time.

Advantages:

- Easy to manage due to its step-by-step nature.
- Well-suited for projects with clear and fixed requirements.

Disadvantages:

- Difficult to make changes once a phase is completed.
- Not flexible for evolving requirements.

Difference between Coding and Unit Testing (5 marks)

Aspect	Coding	Unit Testing
Definition	Writing source code to implement features.	Testing individual units/modules of code.
Purpose	To create a working software system.	To verify each function/module works correctly.
Who Does It	Software developers.	Usually developers or testers.
Tools Used	Programming languages (Java, Python, etc.).	Testing frameworks (JUnit, NUnit, pytest).
When Done	After design phase.	Immediately after coding each unit.

Summary:
Coding brings the design to life, while unit testing ensures that each small part of the code works as expected before combining them into a complete system.