# Object Oriented Software Engineering

MCA Semester III

Department of Computer Science

Gujarat University

# Unit – 5 Software Testing

- Software testing is a critical process in software development that aims to identify defects and ensure that the software meets specified requirements.
- It involves executing a program or application with the intent of finding errors or verifying that it behaves as expected.
- Testing helps to improve the quality of software products by ensuring they are reliable, functional, and user-friendly.

# Software Testing

- For example, consider a banking application that allows users to transfer money.
- Testing would involve checking various scenarios, such as transferring funds between accounts, ensuring that the correct amount is debited and credited, and verifying that no unauthorized transactions occur.
- This ensures that the application functions correctly under real-world conditions.

## Verification and Validation

- Verification and validation (V&V) are two fundamental aspects of software testing that ensure the software product is built correctly and meets user needs.
- Verification focuses on evaluating whether the software meets specified requirements at various stages of development.
- For instance, during the design phase of a project, verification might involve reviewing design documents to ensure they align with user requirements.
- This could include checking whether all necessary features have been included in the design specifications.

## Software Verification Techniques

- Software verification techniques are essential for ensuring that software meets its specifications before it is released. Some common techniques include static analysis, formal methods, and reviews or inspections.
- Static analysis involves examining code without executing it to identify potential errors or vulnerabilities. Tools like SonarQube can analyze code quality by checking for coding standards violations or security vulnerabilities without running the program.

## Software Verification Techniques

- Formal methods use mathematical techniques to prove that a program adheres to its specifications. For example, model checking can systematically explore all possible states of a system model to verify properties like safety or liveness.
- Reviews and inspections involve manual examination of documents and code by peers or experts. This could include code walkthroughs where team members review each other's code for potential issues before merging changes into the main codebase.

## Testing Strategies

- Testing strategies provide a framework for how testing should be conducted throughout the software development lifecycle. Two primary strategies are black-box testing and white-box testing.
- Black-box testing treats the software as a "black box," meaning testers focus on inputs and expected outputs without any knowledge of internal workings.
- For instance, when testing an e-commerce website's checkout process, testers would input various data (like credit card information) and verify if they receive correct confirmation messages without knowing how those processes are implemented in code.

## Testing Strategies

- In contrast, white-box testing involves understanding the internal logic and structure of the code. Testers write test cases based on code paths and conditions.
- For example, if there’s an if-statement in a function determining eligibility for a discount, white-box testers would create tests to cover all possible branches (true/false) of that condition.
- Both strategies are essential; black-box testing ensures functionality from an end-user perspective while white-box testing ensures internal correctness.

## Functional Testing

- Functional testing is designed to verify that each function of the software application operates in conformance with the requirement specification. The main goal is to check whether the software behaves as expected when subjected to various inputs.
- For example, in a customer relationship management (CRM) system, functional testing would involve checking whether users can successfully create new customer records, edit existing records, and delete records as per defined functionalities.

## Structural Testing

- Structural testing focuses on verifying internal structures or workings of a program rather than its functionality. It aims to ensure that all parts of the code are executed at least once during testing.
- Code coverage metrics are often used in structural testing to determine which parts of the program were tested. For instance, if a method has multiple branches (like if-else statements), structural tests would ensure that both branches are executed during tests.

## Structural Testing

- Path testing is another technique used in structural testing where specific paths through the code are tested to ensure they work as intended.
- For example, if there’s a function that calculates discounts based on customer status (e.g., regular vs. premium), path testing would involve creating test cases for both customer types to ensure both paths yield correct results.

## Class Testing

- Class testing is specifically designed for object-oriented programming environments where classes are central components of design. The goal is to validate individual classes in isolation before they interact with other classes.
- In class testing, methods within a class are tested independently from other classes. For example, consider a BankAccount class with methods like deposit(), withdraw(), and getBalance().
- Each method should be tested separately to ensure they perform their intended functions correctly under various conditions (e.g., withdrawing more than balance).

## Class Testing

- Additionally, constructors and destructors should also be tested to confirm proper initialization and cleanup of resources.
- For instance, when creating an instance of BankAccount, tests should check if initial values like balance are set correctly.
- Class testing is crucial because it allows developers to catch errors early before integrating classes into larger systems where interactions can complicate debugging.

## State-Based Testing

- State-based testing focuses on validating system behavior based on different states it can be in during execution. This approach is particularly useful for applications where state transitions dictate functionality.
- For example, consider an online shopping cart application where items can be added or removed from a cart based on user actions. The states might include "empty," "contains items," or "checkout."
- State-based tests would verify that transitions between these states occur correctly when users perform actions like adding or removing items.

## Mutation Testing

- Mutation testing is an advanced technique used to evaluate the effectiveness of test cases by introducing small changes (mutations) into the source code. The idea is that if your test suite is robust enough, it should catch these mutations by failing when they occur.
- For instance, if you have a function that adds two numbers together but you introduce a mutation that changes addition (+) to subtraction (-), effective tests should fail because the output will differ from what's expected based on original functionality.

## Levels of Testing

- Software testing occurs at multiple levels throughout development—each level serves different purposes and focuses on different aspects of quality assurance.
- **Unit Testing**: This level involves testing individual components or functions in isolation from others. For example, if you have a function calculating taxes within an accounting application, unit tests would validate this function independently using various input scenarios.

## Levels of Testing

- **Integration Testing**: Once individual units are verified through unit tests, integration testing checks how these units work together when combined into larger components or systems. An example could be integrating payment processing with order management systems in an e-commerce platform.
- **System Testing**: This level evaluates the complete system's compliance with specified requirements. It involves end-to-end scenarios covering all features collectively—like checking whether users can successfully browse products, add them to their cart, checkout securely using payment methods while receiving confirmation emails.

## Levels of Testing

- **Acceptance Testing**: Finally, acceptance testing validates whether the system meets business needs from an end-user perspective before deployment—often involving real users who perform tasks within production-like environments.

## Testing OOA and OOD Models

- Object-Oriented Analysis (OOA) focuses on understanding requirements through modeling real-world entities while Object-Oriented Design (OOD) translates these models into actual software architecture consisting of classes and objects.
- Testing OOA models involves validating whether identified objects accurately represent real-world entities needed by users—ensuring completeness regarding attributes and behaviors defined during analysis phases.

## Testing OOA and OOD Models

- For instance, if you’re developing a library management system during OOA you might identify objects like Book, Member, Loan. Each must encapsulate relevant properties such as title/author for Book while ensuring behaviors like borrowing/returning books exist within Member.

## Software Testing Tools

- Software testing tools play an essential role in streamlining processes enhancing efficiency across various stages involved throughout entire development lifecycles!
- JUnit serves as one popular framework utilized extensively within Java environments allowing developers automate unit tests efficiently ensuring correctness across individual components developed consistently over time!

## Software Testing Tools

- Selenium represents another widely-used tool enabling automated web application testing facilitating validation efforts surrounding functionality/ behavior experienced by users interacting directly with respective applications encountered regularly!
- Example: Companies like Amazon use tools such as JUnit for unit testing Java applications and Selenium for automated web application testing. These tools help ensure functionality across various services while maintaining high-quality standards throughout development cycles.

## Assignment -5

- What is software testing, and why is it essential in the software development lifecycle? Provide examples of how testing can impact software quality.
- Compare and contrast black-box testing and white-box testing. In what scenarios would each strategy be most effective? Provide an example for each.
- Discuss the importance of functional testing in software development. Describe two functional testing techniques , provide examples of how they can be applied.
- What is structural testing, and how does it differ from functional testing? Explain code coverage and path testing with relevant examples.
- Discuss the specific challenges associated with testing object-oriented software.
- Explain mutation testing and its purpose in evaluating test suite effectiveness. Provide an example of how a mutation might be introduced and how it would be tested.

# Assignment -5

- Explain the concept of an SCM repository. What types of artifacts are typically stored in a repository, and how does version control facilitate collaboration among team members?
- Describe the phases of the SCM process. How do these phases contribute to effective change management in software projects?
- Identify and explain the different types of software risks (technical, project management, and business risks).
- Discuss various techniques for risk identification in software projects. How can tools like SWOT analysis and checklists be utilized effectively during this phase?
- What is an RMMM (Risk Management, Monitoring, and Mitigation) plan? Discuss its key components and how it helps in managing risks throughout the project lifecycle.