CH: 2 Basics of Software Testing, Types of Software Testing, Verification and Validation

2.1 Introduction to Software Testing

What is Software Testing?

• Software Testing is the process of checking a software application to ensure that it works according to requirements
• It helps to find errors (bugs) before the software is delivered to users
• Testing verifies whether the actual output matches the expected output

Simple definition:
Software testing means checking the quality and correctness of software before it is used by customers.

Why is Software Testing Needed?

• To identify defects early
• To improve software quality
• To ensure customer satisfaction
• To reduce cost of fixing errors later
• To make software reliable and secure

Real-life example:
If a calculator app gives wrong answers, users will stop trusting it. Testing ensures correctness.

2.2 Software faults and failures

Bug

Definition

- A bug is an incorrect piece of code in the program
- It is the result of an error

Where it occurs

- During coding
- Inside source code

Example

Total = price + quantity (wrong logic)

Total = price * quantity (correct logic)

-> Bug is inside the code

Error

Definition

- An error is a mistake made by a human
- It usually occurs during requirement analysis, design, or coding

Causes of Error

- Lack of knowledge
- Misunderstanding requirements

- Wrong logic
- Carelessness

Example

- Developer writes + instead of * in a calculation
- Designer misunderstands client requirement

➡ Error is not in the software, it is in human action

Defect

Definition

- A defect is a difference between expected output and actual output
- Identified during testing

When it occurs

- When software is executed
- During testing phase

Example

- Expected output: Total = 100
- Actual output: Total = 60

➡ This mismatch is called a defect

Fault

Definition

- A fault is a cause of incorrect behavior in software
- It is a condition in the software that may lead to failure

Key point

- Every bug is a fault
- Fault exists even if software has not failed yet

Example

- Incorrect condition in if statement
- Wrong database query

➡ Fault is a technical problem inside the system

Failure

Definition

- A failure occurs when software does not perform its intended function
- Seen by end users

When failure occurs

- When a defect reaches the user
- During real execution

Example

- Login button does not work
- App crashes after clicking submit

➡ Failure is visible to users

2.3 Testing Artifacts

• Testing artifacts are documents and tools created during the software testing process

• They help in planning, executing, tracking, and reporting testing activities

• Testing artifacts act as proof that testing was performed properly

Simple definition:

Testing artifacts are records and resources used by testers to ensure software quality.

1) Test Case

Definition

• A test case is a set of steps used to verify a specific function of software

• It defines what to test, how to test, and expected result

Components of a Test Case

Field	Description
Test Case ID	TC_01
Description	Verify login with valid data
Input	Username, Password

Expected Result	Login successful
Actual Result	Login successful
Status	Pass

2) Test Script

Definition

- A test script is a set of instructions written in code to automate testing
- Mainly used in automated testing

Key Points

- Written using testing tools
- Saves time for repetitive tests
- Executes test cases automatically

Difference from Test Case

- Test case → Manual, document-based
- Test script → Automated, code-based

Simple example:

- Script automatically checks login 100 times with different data

3) Test Plan

Definition

- A test plan is a detailed document that describes the testing strategy
- It answers what, when, how, and who will test the software

Contents of a Test Plan

- Objectives of testing
- Scope (what to test, what not to test)
- Test strategy
- Resources required

4) Test Harness

Definition

- A test harness is a collection of software tools and test data
- Used to execute and control automated tests

Components

- Test scripts
- Test data
- Drivers and stubs
- Execution tools

Purpose

- Runs test scripts
- Collects test results
- Helps in automated testing environment

5) Test Suite

Definition

- A test suite is a group of related test cases or test scripts
- Executed together to test a module or system

Example

- Login Test Suite
- Valid login test case
- Invalid password test case
- Empty field test case

Advantages

- Easy to manage testing
- Saves execution time
- Improves test coverage

2.4 Static Testing

Static testing is a software testing technique where the code or documents are reviewed without executing the program.
It helps in finding defects early, saving time and cost.

1. Informal Review

- A casual review with no fixed process
- Performed by peers or team members
- No documentation or checklist required

- Fast and flexible

Example:
A developer asks a teammate to quickly check their code or requirements.

2. Walkthrough

- Conducted by the author of the document/code
- The author explains step by step to reviewers
- Used for knowledge sharing and feedback
- Less formal than inspection

Example:
A developer explains the design document to the team to get suggestions.

3. Technical Review

- Conducted by technical experts
- Focuses on technical correctness and standards
- Follows a structured approach
- Defects are documented

Example:
Senior developers review code for performance, logic, and best practices.

4. Inspection

- The most formal type of static testing
- Follows a strict process and checklist
- Roles are defined (Moderator, Author, Reviewer, Recorder)
- Defects are logged and tracked

Example:
A formal review meeting to inspect a requirement document before approval.

2.5 Dynamic Testing

Dynamic testing is a software testing technique in which the application is executed with test data to identify defects. Unlike static testing, dynamic testing checks the actual behavior of the software during runtime and verifies whether the system works according to the specified requirements. It focuses on validating functionality, performance, usability, and reliability by observing inputs and corresponding outputs.

2.6 Test Levels

Test levels refer to the different stages of testing carried out during the software development lifecycle.

Unit Testing

Unit testing is the first level of testing where individual components or modules of the software are tested separately. It is mainly performed by developers to verify that each unit works correctly according to the design.

Key points of unit testing include:

- Testing is done on small pieces of code such as functions or methods
- Helps in identifying bugs at an early stage
- Usually performed using test cases and test scripts
- Focuses on internal logic and code correctness

Integration Testing

Integration testing is carried out after unit testing to verify the interaction between integrated modules. The goal is to identify issues related to data flow, interfaces, and communication between units.

Key points of integration testing include:

- Combines multiple tested units
- Detects interface and interaction defects
- Can be done using approaches like top-down, bottom-up, or hybrid
- Ensures modules work together as expected

System Testing

System testing is performed on the complete and fully integrated system to ensure it meets the specified requirements. This testing is usually conducted by testers in an environment similar to production.

Key points of system testing include:

- Tests the entire system as a whole
- Verifies functional and non-functional requirements
- Includes performance, security, usability, and reliability testing
- Ensures the system behaves correctly end-to-end

Acceptance Testing

Acceptance testing is the final level of testing conducted to determine whether the software is ready for release and acceptable to the end users. It validates the system against business requirements.

Key points of acceptance testing include:

- Performed by clients or end users
- Focuses on real-world business scenarios
- Confirms whether requirements are fulfilled
- Decides approval or rejection of the product

Techniques of Software Testing

Software testing techniques are the methods used to design test cases and identify defects in software.

2.7 Black Box Testing

Black box testing is a software testing technique in which the tester evaluates the functionality of the application without knowing the internal code or logic. The focus is on verifying whether the system produces correct outputs for given inputs, based on specified requirements and user expectations.

Equivalence Partitioning

Equivalence partitioning is a test case design technique where the input data is divided into valid and invalid partitions. Testing one value from each partition is considered sufficient, as all values in the same partition are expected to behave similarly.

Key points of equivalence partitioning include:

- Reduces the number of test cases
- Divides inputs into equivalent groups
- Covers both valid and invalid inputs
- Improves test efficiency

Boundary Value Analysis (Boundary Data Analysis)

Boundary value analysis focuses on testing the boundary values of input ranges, as defects are most likely to occur at the edges of allowed limits rather than in the middle.

Key points of boundary value analysis include:

- Tests minimum and maximum values
- Includes values just inside and outside boundaries
- Complements equivalence partitioning
- Effective for range-based inputs

Decision Table Testing

Decision table testing is used when the system behavior depends on multiple conditions and their combinations. It represents different conditions and corresponding actions in a table format to ensure complete coverage.

Key points of decision table testing include:

- Suitable for complex business rules
- Covers all possible condition combinations
- Helps avoid missing test scenarios
- Improves logical accuracy

State Transition Testing

State transition testing is applied when the system changes behavior based on different states and events. It verifies valid and invalid transitions between states.

Key points of state transition testing include:

- Tests state changes of the system
- Focuses on events and transitions
- Identifies invalid state transitions
- Useful for workflow-based systems

2.8 White Box Testing

White box testing is a software testing technique in which the tester has full knowledge of the internal structure, logic, and source code of the application. This technique focuses on verifying internal paths, conditions, and statements to ensure that the code works correctly and efficiently. White box testing is mainly performed at the unit and integration testing levels.

Statement Testing and Coverage

Statement testing ensures that each executable statement in the program is executed at least once during testing. The goal is to identify errors in individual lines of code and confirm that no part of the code remains untested.

Key points of statement testing and coverage include:

- Checks execution of every statement

- Measures percentage of statements executed
- Helps find unreachable or dead code
- Simple and easy to implement

Decision Testing and Coverage

Decision testing focuses on verifying that each decision point (such as if–else or switch statements) is executed for all possible outcomes. Decision coverage ensures that both true and false outcomes of each decision are tested.

Key points of decision testing and coverage include:

- Tests all decision outcomes (true/false)
- Identifies logical errors in conditions
- Provides better coverage than statement testing
- Useful for complex decision logic

2.9 Grey Box Testing

Grey box testing is a software testing technique that combines features of both black box testing and white box testing. In this approach, the tester has partial knowledge of the internal structure or code of the application, such as high-level design, database schema, or algorithms, but does not have full access to the source code. This helps the tester design more effective test cases while still validating the system from a user's perspective.

Key points of grey box testing include:

- Tester has limited or partial knowledge of internal logic
- Combines functional and structural testing approaches
- Helps identify defects related to data flow and integration
- More effective than pure black box testing
- Commonly used in integration and system testing

2.10 Non-Functional Testing

Non-functional testing is a type of software testing that focuses on how well the system performs rather than what it does. It checks quality attributes such as performance, reliability, usability, and security to ensure the application provides a smooth, safe, and efficient user experience under real-world conditions.

Performance Testing

Performance testing evaluates how the application behaves in terms of speed, response time, and stability under normal workload conditions.

Key points with a real-life example:

- Measures response time and throughput
- Identifies performance bottlenecks
- Ensures the system meets speed requirements
- *Example:* Checking how fast a shopping app loads product pages when many users browse simultaneously

Stress Testing

Stress testing checks the system's behavior when it is pushed beyond its normal capacity to find the breaking point and observe how it recovers from failure.

Key points with a real-life example:

- Tests extreme workload conditions
- Identifies system failure points
- Checks recovery after crash
- *Example:* Simulating millions of users trying to book train tickets during a festival sale

Load Testing

Load testing verifies how the application performs under an expected number of users or transactions to ensure stability during peak usage.

Key points with a real-life example:

- Tests normal and peak user load
- Ensures system stability
- Helps plan capacity
- *Example:* Testing an online exam portal with 10,000 students logging in at the same time

Usability Testing

Usability testing checks how easy and user-friendly the application is for end users, focusing on navigation, layout, and overall experience.

Key points with a real-life example:

- Evaluates ease of use
- Focuses on user experience
- Identifies confusing interfaces
- *Example:* Observing new users while they try to order food on a delivery app

Security Testing

Security testing ensures that the application is protected against unauthorized access, data breaches, and vulnerabilities.

Key points with a real-life example:

- Identifies security risks
- Protects sensitive data
- Ensures authentication and authorization
- *Example:* Testing whether a banking app prevents unauthorized users from accessing another user's account