

CH1

What is Software Testing?

Software Testing is the systematic process of executing a software application to verify that it meets specified requirements and to identify bugs, errors, or gaps that could affect its quality.

Why do we need Software Testing?

Software testing is essential because it saves money by catching bugs early, improves security by reducing vulnerabilities, ensures the product meets quality standards, and ultimately boosts customer satisfaction—all of which drive business growth.

Who does Testing?

- Project Lead
- Project Developers
- End Users
- Software Tester
- Team Manager

Software Development Lifecycle (SDLC)

The Software Development Lifecycle (SDLC) is a structured process for designing, developing, and testing high-quality software. It ensures that products meet or exceed customer expectations and are delivered on time and within budget, with clearly defined phases.

Requirement Phase

The Requirement Phase is crucial in the SDLC, where business analysts gather and analyze customer needs based on business requirements, then document them in a requirements specification document (name may vary by organization).

Analysis Phase

In the Analysis Phase, product requirements are defined, documented in the SRS (Software Requirement Specification) document, and approved by the customer. This document outlines all the requirements for design and development during the project lifecycle.

Design Phase

In the Design Phase, High-Level Design (HLD) outlines the software's architecture and is created by architects and senior developers, while Low-Level Design (LLD) details the functioning of individual features and components. The resulting High-Level and Low-Level documents serve as inputs for the next phase.

Development Phase

Developers of all levels (seniors, juniors, freshers) are involved in this phase. This is the phase where you start building the code for the software.

Testing Phase

In the Testing Phase, the software undergoes thorough testing—either manually or with automated tools—to identify and fix defects in all components. Once it is error-free, it moves on to the Implementation stage.

Deployment & Maintenance Phase

In the Deployment & Maintenance Phase, the error-free product is delivered to the customer by deployment engineers. As customers use the system, any issues that arise are addressed during the maintenance phase.

Principles of Software Testing

Software testing uncovers bugs that may be missed during development, though it can't guarantee an entirely error-free product. Because it's impractical to test every possible scenario, starting early is crucial to optimize time. Most defects tend to cluster in a few modules, and testing methods must be tailored to the software's specific context. Ultimately, completely bug-free software remains a myth.

CH2

Types of Testing

- Manual
- Automation

What is Manual Testing?

Manual testing is a process where testers execute predefined test cases by hand, without automation tools. It involves creating and running tests to identify issues and compiling a final report, but it can be time-consuming and prone to human error.

Why Need Manual Testing?

Manual testing is important because it:

- **Ensures Bug-free and Stable Software:** It confirms the application meets requirements and delivers a reliable product.
- **Improves Product Familiarity:** Testers gain an end-user perspective, enabling them to write more effective test cases.
- **Verifies Defect Fixes:** It checks that developers have properly fixed issues and that retesting is conducted.

Steps in Manual Testing

- **Requirement Analysis:** Review documentation, guides, and SRS to understand the Application Under Test (AUT).
- **Test Plan Creation:** Develop a plan that covers all requirements.
- **Test Case Creation:** Design test cases based on the requirements.
- **Test Case Execution:** Validate test cases with the team/client and execute them on the AUT.
- **Defect Logging:** Identify and report bugs to developers.
- **Defect Fix and Re-verification:** Retest fixed defects to ensure they pass.

How to perform Manual Testing?

- **Review Documents:** Observe software documents to identify testing areas.
- **Analyze Requirements:** Ensure all customer requirements are covered.
- **Develop Test Cases:** Create test cases based on the requirements.
- **Execute Tests:** Manually run test cases using both black-box and white-box methods.
- **Report Bugs:** Log issues for the development team.
- **Retest:** Verify fixes through retesting.

What is Automation testing?

Automation testing uses specialized tools to automatically execute test cases, completing each testing round much faster than manual methods. This reduced turnaround time helps ensure a positive customer experience and user-friendliness.

What are the levels of Software Testing?

Software testing is performed at several levels corresponding to different SDLC phases to ensure thorough quality checks without overlap. The main levels are:

- Unit Testing – examines individual components.
- Integration Testing – tests interactions between modules.
- System Testing – validates the complete, integrated system.
- Acceptance Testing – confirms the software meets user requirements.

Level 1: Unit Testing

Unit testing is the first testing level that examines individual components. It starts with at least one module, involves testing both positive and negative values, avoids over-testing and assumptions, and stops when maximum coverage is reached.

Level 2: Integration Testing

Integration testing focuses on testing the interaction between integrated components or modules to identify defects in their interactions. for a couple of seconds

Integration testing involves testing groups of modules together to detect defects in their interactions.

Level 3: System Testing

- **End-to-End Testing:** Conducted in an environment mirroring production.
- **Purpose:** Verifies the application as a whole, ensuring both functional and non-functional requirements are met.
- **Types:**
 - *Functional Testing* – Checks if features work as specified.
 - *Performance Testing* – Assesses system responsiveness and stability under load.
 - *Security Testing* – Identifies vulnerabilities.
 - *Usability Testing* – Evaluates user experience.
 - *Compatibility Testing* – Ensures operation across different devices and platforms.
 - *Regression Testing* – Confirms that recent changes haven't adversely affected existing functionality.

Level 4: Acceptance Testing

- **Final Verification:** Ensures the product meets specified requirements.
- **Also Known as:** User Acceptance Testing (UAT), performed by the customer before final approval.
- **Importance:** Identifies minor errors and resolves misunderstandings due to requirement changes or communication gaps.

Software Testing Principles

Follow specific guidelines to ensure the software is defect-free. - Seven Principles:
Addressed step-by-step for a comprehensive testing approach.

Software Testing Principles:

- **Defect Detection:** Testing reveals defects rather than proving their absence.
- **Impossibility of Exhaustive Testing:** Testing every possible scenario is impractical.
- **Early Testing Benefits:** Finding bugs early saves time and money.
- **Defect Clustering:** A small number of modules typically contain most defects.
- **Pesticide Paradox:** Repeating the same tests can eventually become ineffective.
- **Context-Dependent Testing:** Testing approaches should suit the specific project context.
- **No Absolute Quality:** Absence of detected errors doesn't guarantee complete quality.

- **Defects Presence:** Testing reduces defects but never guarantees complete bug-freeness.
- **Exhaustive Testing Impractical:** Testing all input combinations is too costly and time-consuming.
- **Early Testing:** Initiating tests during requirements analysis detects issues early and saves cost.
- **Defect Clustering:** The Pareto Principle shows that 80% of defects come from 20% of modules, highlighting high-risk areas.
- **Pesticide Paradox:** Repeating test cases without updates won't reveal new bugs; regularly revising tests is essential.
- **Context-Dependent Testing:** Testing strategies vary with the software's context (e.g., e-commerce vs. Android apps).
- **Absence-of-Errors Fallacy:** A nearly bug-free product is useless if it fails to meet user requirements.

Types of Software Testing Techniques

- **Static Testing:** Involves examining the software's code or documentation without running it (e.g., code reviews to catch errors early).
- **Dynamic Testing:** Entails executing the software to find bugs and assess performance using manual or automated tests (e.g., automated scripts testing login functionality under varied conditions).

Different Software Testing Techniques

- **All-Pair Testing:** Tests all possible pairs of input parameters for broad coverage with fewer test cases.
- **Cause-Effect Technique:** Maps input causes to expected output effects, ensuring each relationship is validated (e.g., income and credit score affecting loan approval).
- **Risk Coverage:** Focuses testing on high-risk areas of the software, such as critical features like payment processing.
- **Statement Coverage:** Ensures every line of code is executed at least once (e.g., testing all lines in an order processing function).
- **Branch Coverage:** Validates every decision point by testing both true and false conditions (e.g., if-else statements).

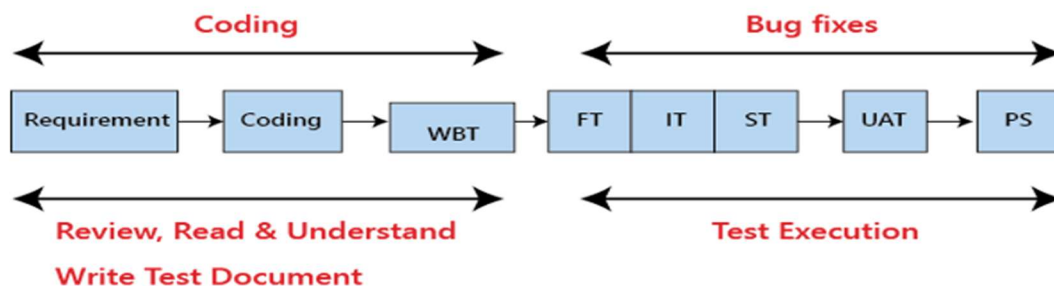
- **Path Coverage:** Confirms that all possible execution paths are exercised, including various loop scenarios.
- **Equivalence Partitioning:** Divides input data into valid and invalid groups to reduce test cases while ensuring coverage (e.g., age input: valid 1–100; invalid <1 or >100).
- **Boundary Value Analysis:** Focuses on testing input limits where errors are likely (e.g., for age 1–100: test values 0, 1, 100, and 101).
- **Decision Table Testing:** Uses a table to map various input combinations to expected outcomes (e.g., testing customer types and purchase amounts in a discount system).
- **State Transition Testing:** Verifies that state changes occur as expected, such as transitioning from “Logged Out” to “Logged in” and handling “Incorrect Password” scenarios.
- **Use Case Testing:** Assesses the software using real-world scenarios (e.g., “Add Item to Cart,” “Checkout”) to ensure it meets user requirements.
- **Error Guessing:** Relies on tester intuition to identify likely defect areas, such as potential issues with special characters in input fields.

CH3

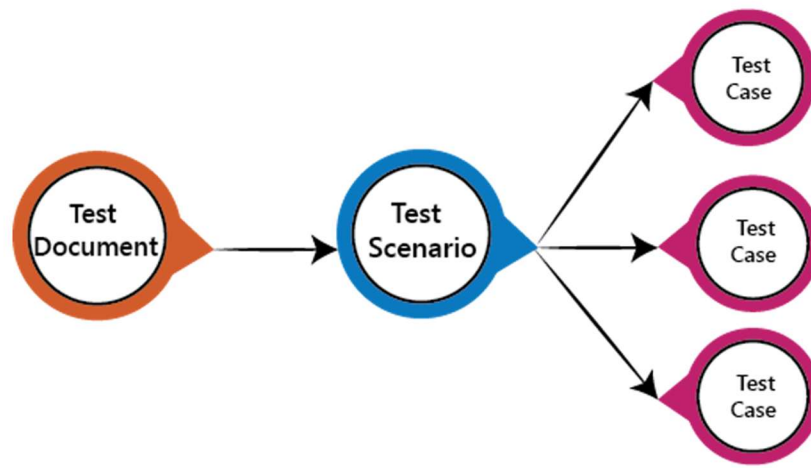
What is Testing Documentation?

Testing Documentation:

- Comprises artifacts created during or before software testing.
- Highlights the importance of processes for customers, individuals, and organizations.
- Saves time, effort, and resources.
- Aims to eliminate doubts about testing activities.



Type of Test Document



Test Scenarios

- Detailed documents of test cases covering end-to-end functionality using linear statements.
- Serve as a high-level classification of testable requirements.
- Grouped by module functionality and derived from use cases.

How to write Test Scenarios?

- Review BRS, SRS, and FRS documents.
- Determine technical aspects and objectives for each requirement.
- Identify user operation methods.
- Consider potential misuse and security risks.
- List scenarios to verify software functions.
- Create a traceability matrix linking scenarios to requirements.
- Review scenarios with project supervisors and stakeholders.

Features of Test Scenario

- Guides testers in sequence.
 - Reduces complexity and avoids repetition.
 - Provides detailed, linear test discussions.
 - Useful when time is limited for writing test cases.
 - Saves time.
-
- Facilitates easy maintenance through independent updates.

Example of Test scenarios

Here we are taking the **Gmail application** and writing a test scenario for a modules which are most commonly used such as **Login**.

- Enter the valid login details (Username, password), and check that the home page is displayed.
- Enter the invalid Username and password and check for the home page.
- Leave Username and password blank, and check for the error message displayed.
- Enter the valid Login, and click on the cancel, and check for the fields reset.
- Enter invalid Login, more than three times, and check that account blocked.
- Enter valid Login, and check that the **Username** is displayed on the home screen.

Test Case

- Derived from test scenarios as a first-level action.
 - Defines a format to verify if a software/module functions correctly.
 - Focuses on "what to test" and "how to test."
-
- Contains detailed parameters, requiring significant documentation, resources, and time for execution.

When do we Write Test Cases?

- **Before Development:** Write test cases prior to coding to identify product requirements and plan for later testing.
- **After Development:** Develop test cases once features are complete but before launch to verify functionality.

- **During Development:** Create test cases in parallel with coding to test components as they are developed.

Why Write Test Cases?

- Ensures software meets customer expectations.
- Verifies consistency with specified conditions.
- Helps narrow down and manage software updates.
- Enhances test coverage by documenting all scenarios.
- Supports consistent test execution through detailed test cases.
- Aids maintenance with comprehensive documentation.

Test case template

- **Actual vs. Expected:** Typically match; if a test step fails, skip the actual result field and note bugs in comments. Remove the input field and add that info to the description.
- **Header Components:** Includes step number (critical for bug documentation and prioritization), test case type (functional, integration, system, positive, negative, or both), release (which may include multiple versions), pre-conditions, test data (e.g., usernames, passwords, account numbers), and severity (major, minor, or critical based on module features).

Fields	Description
Test Case ID	Each test case should have a unique ID.
Test Case Description	Each test case should have a proper description to let testers know what the test case is about.
Pre-Conditions	Conditions that are required to be satisfied before executing the test case.
Test Steps	Mention all test steps in detail and to be executed from the end-user's perspective.
Test Data	Test data could be used as input for the test cases.
Expected Result	The result is expected after executing the test cases.
Post Condition	Conditions need to be fulfilled when the test cases are successfully executed.
Actual Result	The result that which system shows once the test case is executed.
Status	Set the status as Pass or Fail on the expected result against the actual result.

Fields	Description
Project Name	Name of the project to which the test case belongs.
Module Name	Name of the module to which the test case belongs.
Reference Document	Mention the path of the reference document.
Created By	Name of the tester who created the test cases.
Date of Creation	Date of creation of test cases.
Reviewed By	Name of the tester who reviews the test case.
Date of Review	When the test cases were reviewed.
Executed By	Name of the tester who executed the test case.
Date of Execution	Date when the test cases were executed.

Best Practice for Writing Test Case

- **Clear:** Easy to read.
- **Unique:** Meets client needs.
- **Accurate:** Uses real data only.
- **Traceable:** Keeps clear records.
- **Diverse:** Includes various inputs.
- **Descriptive:** Uses clear names and short descriptions.
- **Comprehensive:** Covers every condition.
- **Consistent:** Always yield the same result.
- **Varied:** Use different testing methods.
- **User-Focused:** Write tests from the end-user's view.
- **Unique IDs:** Assign clear, unique identifiers.
- **Clear Conditions:** List preconditions and post conditions.
- **Reusable:** Ensure tests work after code updates.
- **Precise:** Specify the exact expected outcome.

Formal and Informal Test Case

- **Formal Test Cases:** Follow a structured format with set input data, expected results, and specific steps.

- **Informal Test Cases:** No fixed format; written in real-time without predefined inputs or expected outcomes.

Types of Test Cases

- **Functionality:** Checks interface using black box testing.
- **Unit:** Tests individual parts separately.
- **UI:** Checks user-interface components.
- **Integration:** Tests combined units for smooth operation.
- **Performance:** Measures response time and efficiency.

- **Database:** Tests tables and triggers.
- **Security:** Checks permissions and data safety.
- **UX:** Tests ease of use.
- **User Acceptance:** Testers create, clients review.

What is a QA test case?

- **QA Test Case:** Details conditions to verify if a system or feature works as expected.
- **Purpose:** Ensures software meets requirements.
- **Benefits:** Provides consistency, traceability, and efficiency.

Benefits of writing high-quality test cases

- **More Coverage:** Tests more scenarios.
- **Clearer Communication:** Aligns the team.
- **Consistent Results:** Repeatable testing.
- **Faster Bug Detection:** Spots defects quickly.
- **Easy Automation:** Simplifies automation.
- **Better Quality:** Increases confidence.

Example test cases for a login page

→ Below is an example of preparing various test cases for a login page with a username and password.

- **Unit Test case:** Here we are only checking if the username validates at least for the length of eight characters.
- Here it is only checked whether the passing of input of thirteen characters is valid or not. So, since the character word 'university' is entered then the test is successful it would have failed for any other test case.

Test Id	Test Condition	Test Steps	Test Input	Test Expected Result	Actual Result	Status	Remarks
1.	Check if the username field accepts the input of thirteen characters.	1. Give input	university	Accepts for thirteen characters.	Accepts for thirteen characters.	Pass	None

Example test cases for a login page (Cont.)

- **Functionality Test case:**

- Here it is checked whether the username and password both work together on the login click.
- Here it is being checked whether passing wrong and right inputs and if the login functionality is working or not, its showing login is successful for the right credentials and unsuccessful for the wrong ones, hence both tests have passed otherwise would have failed.

Test Id	Test Condition	Test Steps	Test Input	Test Expected Result	Actual Result	Status	Remarks
1.	Check that with the correct username and password able to log in.	1. Enter the username 2. Enter the password 3. Click on the login	username: university password: universityforever	Login successful	Login successful	Pass	None
2.	Check that if with an incorrect username and password able to not login.	1. Enter the username 2. Enter the password 3. Click on the login	username: universitycue password: universitytogether	Login unsuccessful	Login unsuccessful	Pass	None

Example test cases for a login page (Cont.)

- **User Acceptance Test Case:** Here the user feedback is taken if the login page is loading properly or not.
- Here it is being checked in by clicking on the login button if the page is loaded and the ' **Welcome to login page** ' message is displayed. The test has failed here as the page was not loaded due to a browser compatibility issue; it would have loaded if the test had passed.

Test Id	Test Condition	Test Steps	Test Input	Test Expected Result	Actual Result	Status	Remarks
1.	Check if the loading page loading efficiently for the client.	1. Click on the login button.	None	Welcome to the login page.	Welcome to the login page.	Fail	The login page is not loaded due to a browser compatibility issue on the user's side.