Test Engineering

- "Imagine you download a new app, but every time you try to log in, it crashes. How frustrating would that be?
- This is why software testing is crucial—it ensures that software works as expected before users experience it.“

- Have you ever faced a software bug?
- What do you think happens when software isn't tested properly?

- Define Software Testing
- simple definition:
- "Software testing is the process of evaluating a software application to detect and fix bugs, ensure quality, and verify that it meets requirements."

- why testing is important:

- Ensures Quality – Prevents failures in real-world usage.

- Saves Costs – Fixing bugs early is cheaper than fixing them later.

- Builds User Trust – A well-tested software gives a good user experience.

Fundamentals of Software Testing

- Software testing is the process of verifying and validating a software application to check whether it:
- Meets the specified requirements
- Functions correctly without errors
- Ensures reliability, performance, and security

- Verification vs. Validation:
- Verification: Checking if the software is built correctly (Are we building the product right?)
- Validation: Checking if the right software is built for the user (Are we building the right product?)

Software Testing Life Cycle (STLC)

- 1 Requirement Analysis
- Understanding requirements
- Identifying testable and non-testable requirements

- 2 Test Planning
- Defining test strategy
- Selecting test tools
- Estimating effort & cost

- 3 Test Case Development
- Writing test cases & test scripts
- Creating test data

- 4 Test Environment Setup
- Setting up software, hardware, and network configurations

- 5 Test Execution
- Running test cases
- Logging defects & reporting

- 6 Test Closure
- Preparing test reports
- Reviewing lessons learned

Software Testing Lifecycle (STLC)

- **Requirement Analysis** – Understand what needs to be tested.
  Test Planning – Decide testing strategies and tools.
  Test Case Development – Write test cases.
  Environment Setup – Prepare the testing environment.
  Test Execution – Run tests and report bugs.
  Test Closure – Review test results and improve processes.

Types of Software Testing

- **Functional Testing (Validating software behavior)**

- Unit Testing (Testing individual components)
- Integration Testing (Testing how components work together)
- System Testing (End-to-end testing of the whole system)
- User Acceptance Testing (UAT) (Final validation by users)

- **Non-Functional Testing (Checking software attributes like performance, security, and usability)**

- Performance Testing (Speed, load handling)
- Security Testing (Identifying vulnerabilities)
- Usability Testing (Checking ease of use)
- Compatibility Testing (Ensuring it works across devices & browsers)

- Levels of Testing
- White Box Testing – Internal code structure testing
- Black Box Testing – Testing functionality without looking at the code
- Gray Box Testing – Combination of both white & black box testing

- The two fundamental testing techniques used to evaluate software functionality and internal structure are **Black Box Testing** and **White Box Testing**.

Black Box Testing (Functional Testing)

- **Definition**: Black box testing evaluates the software's functionality without considering its internal code structure or logic. The tester interacts with the application through the user interface and verifies the expected outputs based on given inputs.
- **Focus**: Functional requirements, user interactions, and expected behavior.
- **Who Performs It**: Testers, QA engineers, or end users.

- **Techniques**:
  - **Equivalence Partitioning**: Divides input data into valid and invalid partitions to reduce test cases.
  - **Boundary Value Analysis**: Tests edge cases by checking values at boundaries (e.g., minimum, maximum, just inside/outside boundaries).
  - **Decision Table Testing**: Uses tables to cover different input conditions and their outcomes.
- **Example**:
  - A login page requires a username and password. A black box tester would test:
    - Correct login credentials → Successful login.
    - Incorrect password → Error message.
    - Blank fields → Validation error.
  - The tester does not check how the login function is coded, only the outcomes.

White Box Testing (Structural Testing)

- **Definition**: White box testing evaluates the internal structure, logic, and code implementation of the software. The tester has full knowledge of the code and tests the system at the code level.
- **Focus**: Code logic, security vulnerabilities, paths, and internal processing.
- **Who Performs It**: Developers or testers with programming knowledge.

- **Techniques**: **Statement Coverage**: Ensures every line of code is executed at least once.
- **Branch Coverage**: Tests all possible paths, including if-else and switch conditions.
- **Path Coverage**: Ensures all possible execution paths are tested

- **Example**: Suppose there is a function to calculate discounts

```
def calculate_discount(price, customer_type):
    if customer_type == "VIP":
        return price * 0.8  # 20% discount
    else:
        return price * 0.9  # 10% discount
```

- White box testing would verify:
- Whether all conditions (VIP and non-VIP) are covered.
- Whether all code statements execute correctly.
- If any unhandled exceptions exist.

Key Differences

Feature	Black Box Testing	White Box Testing
Focus	Functionality	Code structure
Knowledge Required	No knowledge of code	Deep understanding of code
Performed By	Testers, users	Developers, testers with coding skills
Techniques	Equivalence partitioning, boundary testing, decision tables	Statement coverage, branch coverage, path testing
Example	Testing login functionality	Checking all code execution paths in the login function

Both techniques complement each other: **Black box testing ensures functional correctness, while white box testing ensures code reliability and security.**

Black Box vs. White Box Testing

Black Box Testing and **White Box Testing** are two fundamental approaches in software testing.

Criteria	Black Box Testing	White Box Testing
Definition	Tests the system **without knowing** internal code.	Tests the system **with full knowledge** of the internal code.
Focus	Functionality & behavior of the software.	Internal structure, code logic, and paths.
Performed By	QA testers, end-users.	Developers, security testers.
Testing Basis	Based on **requirements & specifications**.	Based on **code structure & logic**.
Access to Code	❌ No access to source code.	✅ Full access to source code.
Types	Functional Testing, System Testing, Acceptance Testing.	Unit Testing, Integration Testing, Security Testing.
Example	Testing an **e-commerce checkout process** (Does payment work?).	Testing **if-else conditions** in the checkout function.
Tools	Selenium, Postman, JMeter.	JUnit, PyTest, SonarQube.

- **Black Box Testing (Functional Testing)**
- Focuses on **what the software does**, NOT how it works.
- Tester provides inputs → checks if the expected output matches.
- **Example:**
- Testing a **login page**:
  ✓ **Input:** Enter correct username & password.
  ✓ **Expected Output:** User logs in successfully.
  ✓ **Input:** Enter wrong password.
  ✓ **Expected Output:** Error message appears.

- **Common Black Box Testing Techniques:**
- ✓ **Equivalence Partitioning** – Group similar inputs together.
  ✓ **Boundary Value Analysis** – Test minimum & maximum values.
  ✓ **Error Guessing** – Predict common mistakes (e.g., leaving a field empty).
- **Tools:**
- ✅ Selenium (UI Testing)
  ✅ JMeter (Performance Testing)
  ✅ Postman (API Testing)

- **White Box Testing (Code-Based Testing)**
- Focuses on **how the code works** internally.
- Checks logic, loops, security vulnerabilities, and performance issues.
- **Example:**
- Testing the **payment function** in the backend:
  ✓ Check if **all if-else conditions** are covered.
  ✓ Verify if **loops terminate correctly**.
  ✓ Detect security issues like **SQL injection**.

- **Common White Box Testing Techniques:**
- ✓ **Statement Coverage** – Ensure every line of code runs at least once.
  ✓ **Branch Coverage** – Test all possible code paths.
  ✓ **Path Coverage** – Cover all execution flows.
- **Tools:**
- ✓ JUnit (Java)
  ✓ PyTest (Python)
  ✓ SonarQube (Code Quality Analysis)

- **Gray Box Testing (Hybrid Approach)**
- ◆ A mix of **Black Box + White Box** testing.
  ◆ **Who performs it?** – Testers with **partial knowledge** of the system.
  ◆ **Example:** API Testing (knowing the request-response format but not the internal implementation).

✅ **Black Box Testing** → Tests **functional behavior** without knowing the code.

✅ **White Box Testing** → Tests **internal code structure** & logic.

✅ **Gray Box Testing** → Combines **both approaches**.

Principles of Software Testing

- Testing shows the presence of defects, not their absence
- Exhaustive testing is impossible (We can’t test everything!)
- Early testing saves time & cost
- Defects cluster in certain areas of the software
- The pesticide paradox (Reusing the same tests won't find new bugs)
- Testing is context-dependent (Different software needs different tests)
- Absence of errors is a fallacy (A bug-free product may still not meet user needs)

Manual vs. Automation Testing

- Manual Testing: Performed by testers manually (Good for exploratory, usability & ad-hoc testing)
- Automation Testing: Uses scripts/tools (Good for repetitive & regression tests)

- Popular Automation Tools:
- ✓ Selenium (Web automation)
- ✓ Cypress (Web testing)
- ✓ Appium (Mobile automation)
- ✓ JMeter (Performance testing)

- **Bug Life Cycle**
- Every defect found in testing goes through the following stages:
  - **New** → **Assigned** → **In Progress** → **Fixed** → **Retested** → **Closed**

- **Importance of Software Testing**
- ✓ Improves **software quality**
  ✓ Detects **bugs early** to save time & cost
  ✓ Ensures **security & reliability**
  ✓ Enhances **user experience**

Different Levels of Software Testing

- Software testing is conducted at **multiple levels** to ensure the quality, functionality, and performance of an application. The main **four levels of testing** are:
- 1 **Unit Testing** (Testing individual components)
  2 **Integration Testing** (Testing interactions between components)
  3 **System Testing** (Testing the entire application)
  4 **Acceptance Testing** (Validating with end-users or clients)

- Unit Testing (Lowest Level of Testing)
- What is it?
- Tests individual components (functions, classes, modules).
- Ensures that each unit works independently.
- Performed by developers before integration.

- **What is Unit Testing?**
- **Unit Testing** is a type of **software testing** where individual components or modules of an application are tested in **isolation** to ensure they function correctly.
- ✓ It is performed **by developers** or testers during the **development phase**.
  ✓ It helps in **early bug detection**, reducing cost & effort in later stages.

- **Why is Unit Testing Important?**
- ✓ Ensures each module works **independently**
  ✓ Catches **bugs early** in development
  ✓ Makes **debugging easier**
  ✓ Improves **code quality** and maintainability
  ✓ Helps in **refactoring & scalability**

- **Unit Testing Process**
- 1 **Write a Test Case** – Define inputs & expected outputs.
  2 **Execute the Test** – Run the test to check if the function works.
  3 **Check Results** – Compare actual vs. expected output.
  4 **Fix Bugs** – If the test fails, fix the function & retest.
  5 **Repeat** – Keep testing after every small change.

Unit Testing Frameworks & Tools

Programming Language	Unit Testing Framework
JavaScript	Jest, Mocha, Jasmine
Python	unittest, PyTest
Java	JUnit, TestNG
C#	NUnit, MSTest
PHP	PHPUnit
Ruby	RSpec

- Testing a **login function** to check if it correctly validates user credentials.

Testing Type	Purpose
Unit Testing	Tests individual components (functions, methods)
Integration Testing	Tests interaction between modules
System Testing	Tests the entire application
Acceptance Testing	Validates from the end-user perspective

- **Challenges in Unit Testing**
- ⚠ Difficult for **legacy code** without modularity
  ⚠ Hard to test **UI components**
  ⚠ Time-consuming **if not automated**

- **Integration Testing**
- Tests how different **modules/components** interact with each other.
- Detects issues in **data flow & API communication**.
- Performed **after unit testing**.

- **Example:**
- Testing if a **payment gateway** correctly integrates with an **order system**.

Types of Integration Testing:

- Integration Testing ensures that different modules or components of a software system work together correctly. It comes in different types based on how the components are combined and tested
- **Top-Down Testing:** Test **higher-level** modules first.
  Bottom-Up Testing: Test **low-level** modules first.
  Big Bang Testing: Test **all modules at once** (risky).
- **Incremental Testing:** Test **one module at a time**.
- **Tools:**

Postman (API Testing)
SoapUI (Web Services Testing)
JUnit (Java)

Big Bang Integration Testing

- 🔷 All modules are integrated simultaneously, and the entire system is tested as a whole.
  ✅ **Pros**: Simple and saves time in small projects.
  ❌ **Cons**: Difficult to pinpoint errors, as everything is tested at once.
- 💡 **Example:** Testing a complete e-commerce system after integrating the cart, payment, and order modules all at once.

Top-Down Integration Testing

🔽 Testing starts with the high-level modules and progresses down to the lower-level modules.
  🔽 Uses **stubs** (dummy modules) to simulate missing components.

✅ **Pros**: Helps in early detection of design flaws.
❌ **Cons**: Lower-level modules are not tested early.

- 💡 **Example:** Testing a banking app by first integrating the **dashboard module**, then adding the **transaction module**, and finally the **account details module**.

Bottom-Up Integration Testing

🔹 Testing begins with lower-level modules and moves up to higher-level modules.
  🔹 Uses **drivers** (temporary programs) to simulate higher-level components.

✓ **Pros**: Critical components are tested early.
  ✗ **Cons**: GUI-related issues may be discovered late.

- 💡 **Example:** In a hospital management system, first testing the **database module**, then integrating the **patient management system**, and finally linking it with the **hospital dashboard**.

Sandwich (Hybrid) Integration Testing

🔹 A combination of **Top-Down** and **Bottom-Up** approaches.
  🔹 Testing happens at both the top and bottom levels simultaneously, meeting in the middle.

✅ **Pros**: Faster testing with the advantages of both approaches.
❌ **Cons**: Can be complex and resource-intensive.

- 💡 **Example:** In a banking system, testing both the **user interface** and **backend transactions** simultaneously before linking them together.

Incremental Integration Testing

🔹 Modules are tested **one by one** and integrated gradually.
  🔹 Bugs are fixed after each integration step.

✅ **Pros**: Easier to identify defects.
  ❌ **Cons**: Takes longer than Big Bang testing.

- 💡 **Example:** In a messaging app, first testing the **login module**, then adding the **chat feature**, and later integrating the **video call feature** step by step.

- Each integration testing type is useful depending on the project size, complexity, and requirements. **Incremental testing** is commonly preferred as it allows early bug detection.

Types of Integration Testing

- Integration Testing ensures that different modules or components of a software system work together correctly. It comes in different types based on how the components are combined and tested.

- **System Testing**
- Tests the **entire application** as a whole.
- Ensures the system meets functional and non-functional requirements.
- Performed by **QA testers**.

- **Example:**
- Testing an **e-commerce website** to check:
  ✓ Can users add/remove items from the cart?
  ✓ Does the checkout process work correctly?
  ✓ Is the website **responsive** on mobile & desktop?

- **Types of System Testing:**
- ✓ **Functional Testing** – Verifies **business logic**.
  ✓ **Non-Functional Testing** – Tests **performance, security, usability**.

- 🛠 **Tools:**
- ✅ Selenium (Web UI Automation)
  ✅ JMeter (Performance Testing)
  ✅ Appium (Mobile Testing)

- **Acceptance Testing (Highest Level of Testing)**
- Ensures the software meets **business requirements**.
- Performed by **end-users, clients, or stakeholders**.
- **Final step before deployment**.
- ✅ **Types of Acceptance Testing:**
- ✓ **Alpha Testing:** Done **before** releasing to users.
  ✓ **Beta Testing:** Done **after** releasing to a small group of users.
- 🛠 **Example:**
- A retail company tests an **online shopping platform** before launching to customers.
- 🛠 **Tools:**
- ✅ TestRail (Test Case Management)
  ✅ Jira (Bug Tracking)

Summary Table: Levels of Testing

Level	Purpose	Performed By	Tools
Unit Testing	Tests individual components	Developers	JUnit, PyTest, Jest
Integration Testing	Tests how modules interact	Developers, Testers	Postman, SoapUI
System Testing	Tests the whole application	QA Testers	Selenium, JMeter
Acceptance Testing	Validates with real users	Clients, End-users	TestRail, Jira