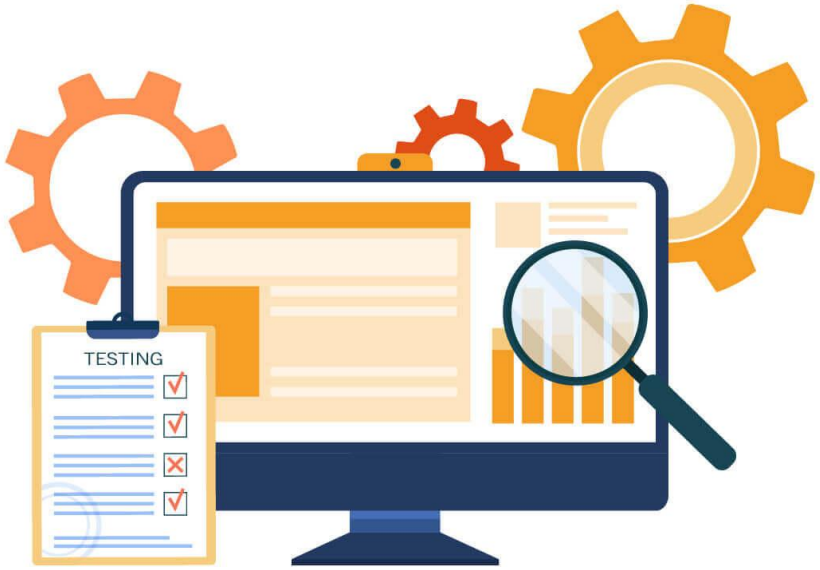
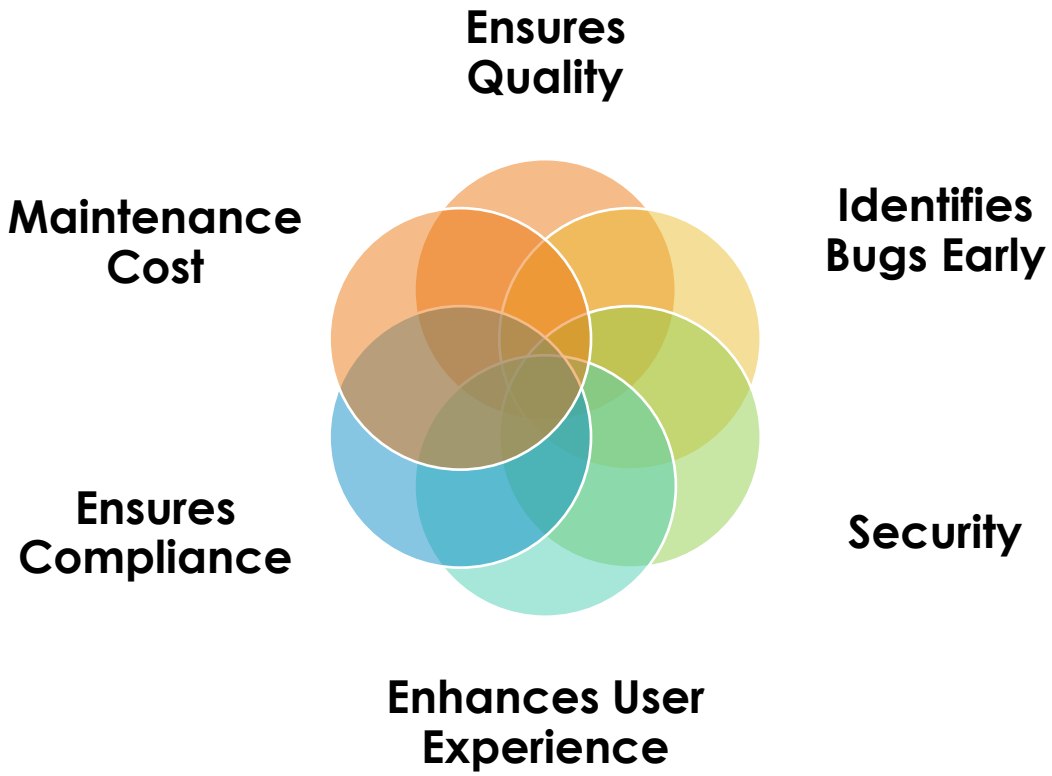
INTRODUCTION TO TESTING



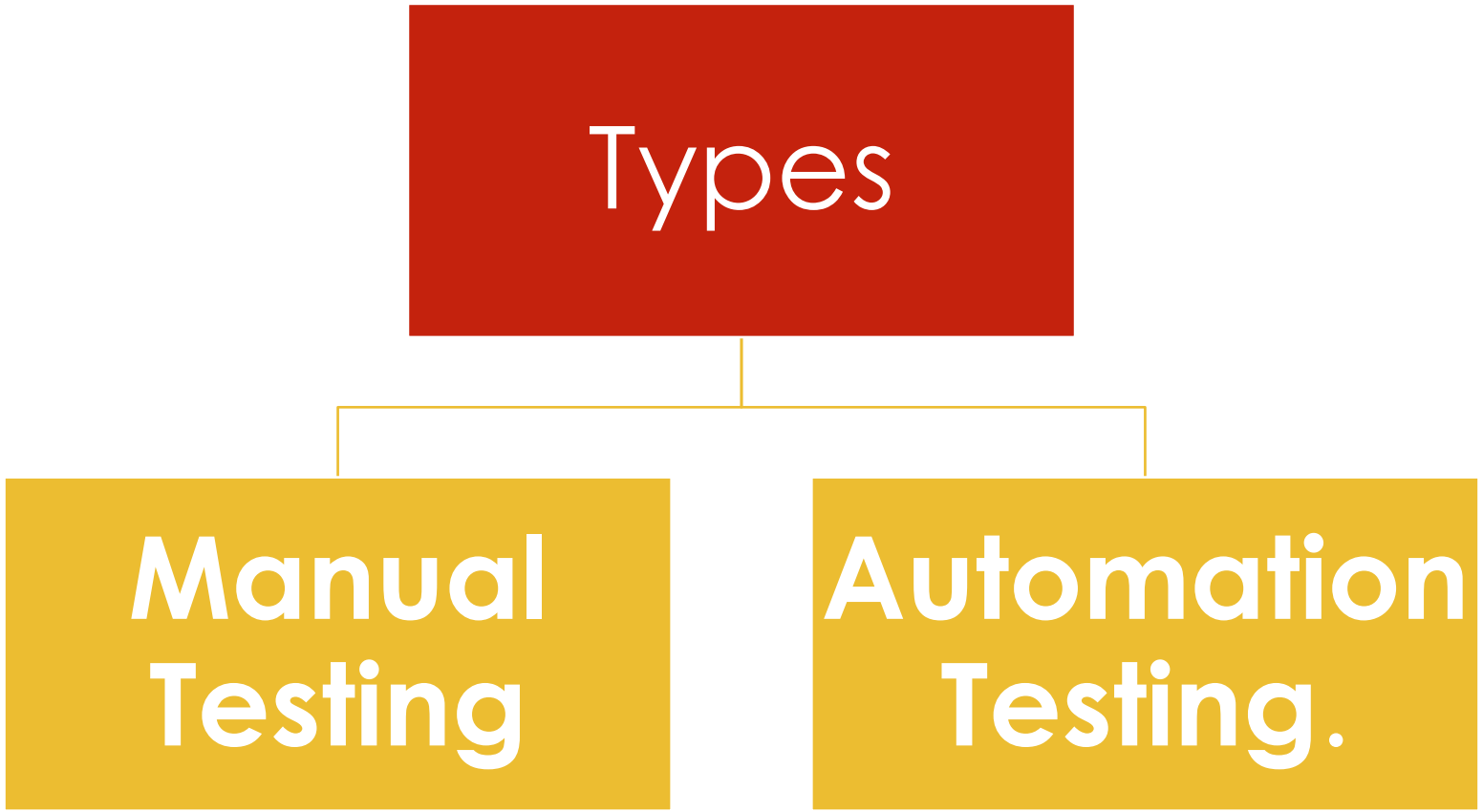
TESTING

- Testing is the process of evaluating a software application to identify any defects, bugs, or errors before it is deployed to end-users.
- The goal of testing is to ensure that the software meets its requirements, functions correctly, and provides a smooth user experience.

WHY TESTING


Ensures Quality
Identifies Bugs Early
Security
Enhances User Experience
Ensures Compliance
Maintenance Cost

TYPES OF TESTING



FUNCTIONAL TESTING

- **Use Case:** Verifying that a login feature works correctly.

Unit Testing	• Tests individual components or functions. • Testing a function in Java that adds two numbers.
Integration Testing	• Tests multiple components working together. • Checking the login page successfully connects to database.
System Testing	• Tests the entire system as a whole. • Testing an e-commerce website after integrating all features.
User Acceptance Testing (UAT)	• Ensures the software meets user needs. • A client testing a mobile banking app before release.

NON-FUNCTIONAL TESTING

- Focuses on aspects like performance, usability, and security.

Performance Testing	Load Testing	Stress Testing	Usability Testing	Security Testing
• Checks speed, responsiveness and stability under load. • Testing how many users an e-commerce website can handle during sales.	• Determines the maximum capacity the system can handle. • Checking how a booking website performs with 10,000 users.	• Tests system behavior under extreme conditions. • Simulating a server crash to see if the system recovers.	• Ensures the software is user-friendly. • Checking how easy it is to navigate an online food delivery app.	• Identifies vulnerabilities and risks. • Testing for SQL injection in a login form.

White Box Testing

- Tests internal code and logic.
- Developers writing unit tests for a function.

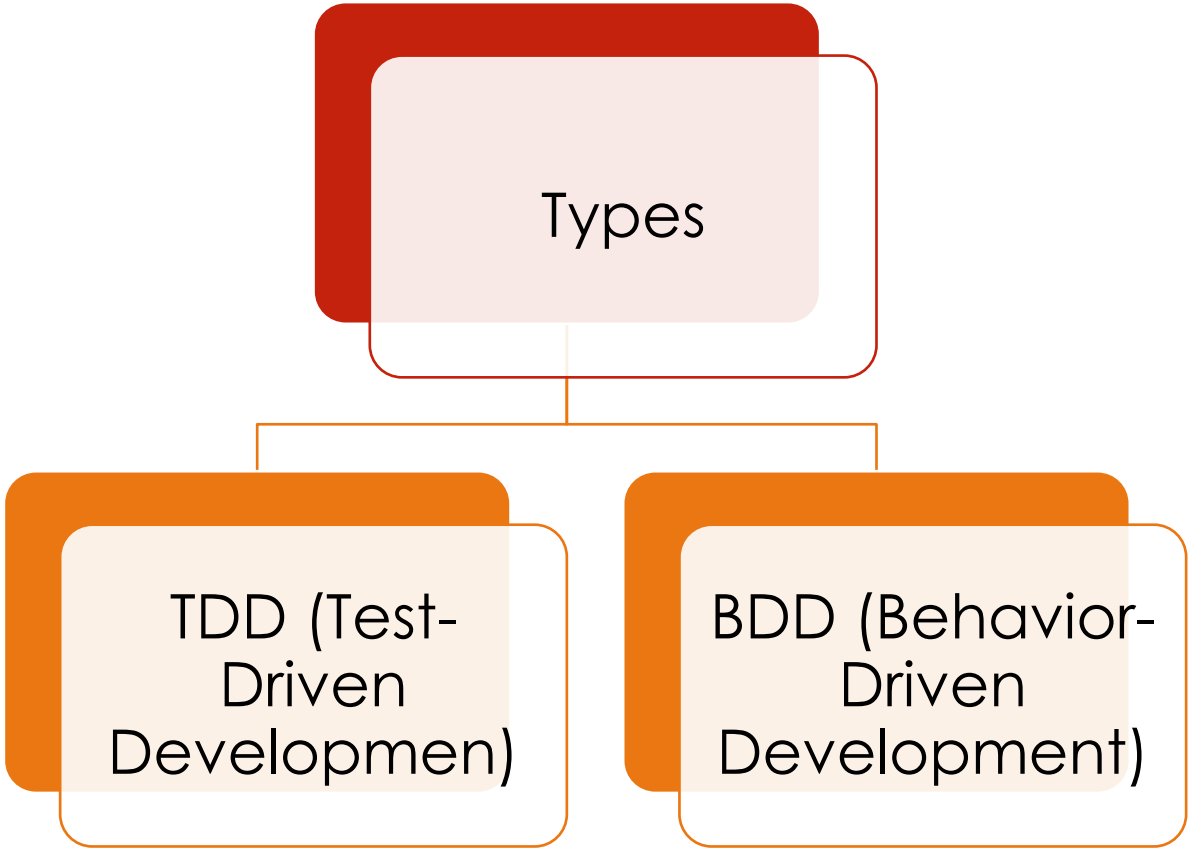
Black Box Testing

- Tests functionality without knowing internal code.
- Tester verifying whether a signup form works without checking the code.

Grey Box Testing

- Combines both approaches.
- Security testing where the tester has limited knowledge of the internal system.

TESTING APPROACH



TEST-DRIVEN DEVELOPMENT (TDD)

- TDD is a **test-first approach** where developers write tests before writing the actual code.
- The process follows the **Red-Green-Refactor** cycle:
  - **Write a failing test** (Red)
  - **Write the minimal code** to make the test pass (Green)
  - **Refactor** the code to improve quality (Refactor)

BDD

- Behavior-Driven Development
- BDD is an extension of TDD but focuses on the **behavior** of the system rather than implementation details.
- It uses natural language syntax (Given-When-Then) to make tests understandable by non-technical stakeholders.
  - Define Feature File
  - Implement Step Definitions
  - Run the BDD Test

TDD VS BDD

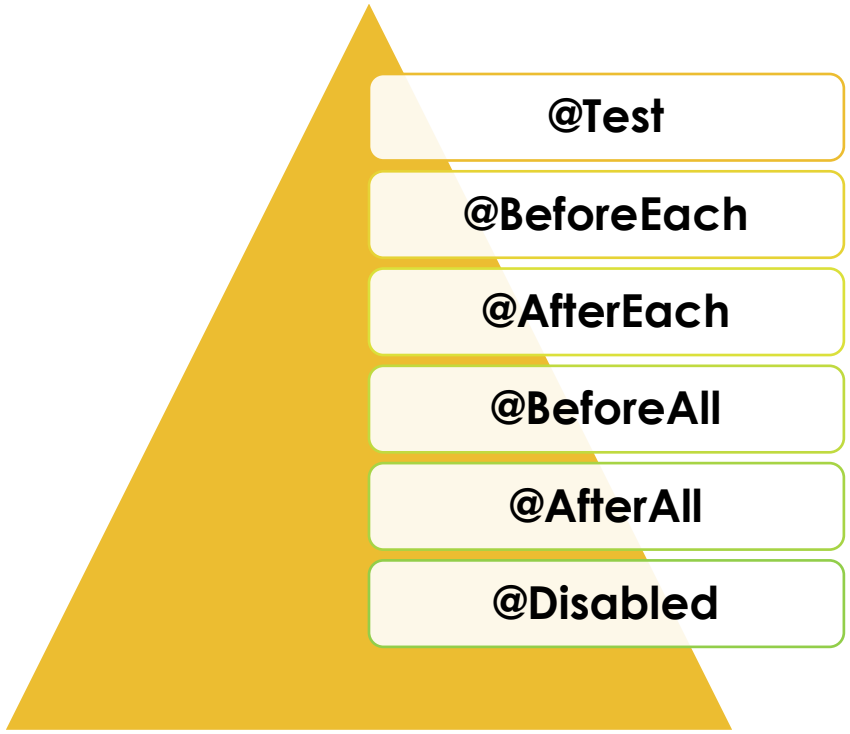
Feature	TDD	BDD
Focus	Code Implementation	User Behaviour
Test Syntax	Code-based (JUnit, Jest)	Natural Language (Cucumber)
Who Uses It?	Developers	Developers, Testers, Business Analysts
Example Style	Unit tests for methods	Given-When-Then for user behavior

JUNIT TESTING

- JUnit is a popular testing framework for Java used to write and run unit tests.
- It allows developers to ensure that individual components of their application work correctly.
- Key Concepts
  - **Annotations**
  - **Assertions**

ANNOTATIONS

- Used to define test cases, setup, teardown, etc.



ASSERTIONS

- **Used to validate test results**
  - assertEquals(expected, actual)
  - assertNotEquals(value1, value2)
  - assertTrue(condition)
  - assertFalse(condition)

LET'S WRITE SOME TEST CASES

- Create JAVA Project using Maven
- Add dependencies for JUNIT
- Write Test Cases

INTRODUCTION TO JMETER

- Apache **JMeter** is an open-source tool used for **performance testing, load testing, and functional testing** of web applications, APIs, databases, and more.
- It allows you to simulate multiple users sending requests to a server and measure its response times, throughput, and behavior under load.

KEY FEATURES

- **Performance Testing** – Simulates multiple concurrent users to analyze system performance.
- **Load Testing** – Measures how a system behaves under different loads.
- **Stress Testing** – Determines the breaking point of an application.
- **API Testing** – Tests REST and SOAP APIs with custom requests.
- **Database Testing** – Tests queries and database performance.
- **GUI and CLI Support** – Can be run through a graphical interface or from the command line.

JMETER FLOW

- JMeter acts as a **virtual user** that sends requests to a system and measures its performance.

- FLOW:
- **Create a Test Plan** – Define the testing scenario.
- **Add Thread Groups** – Simulate multiple users.
- **Configure Samplers** – Define the type of requests (HTTP, JDBC, FTP, etc.).
- **Add Listeners** – Collect test results in different formats.
- **Run the Test** – Execute the test and analyze the reports

WHERE WE CAN USE JMETER?

- Web application load testing
- REST API performance validation
- Database query execution time analysis
- FTP and file upload/download performance testing
- CI/CD pipeline integration for automated testing

INTRODUCTION TO APPIUM

- **Appium** is an open-source test automation tool used for testing **mobile applications** on Android and iOS devices.
- It allows you to write tests using **Selenium WebDriver** and supports multiple programming languages such as **Java, Python, JavaScript, C#, and Ruby**.
- **Appium is best for:**
- **Automating Native Apps**
  - Apps built for a specific platform (Android/iOS).
- **Automating Hybrid Apps**
  - Apps using WebView with native components.
- **Automating Mobile Web Apps**
  - Web apps accessed via mobile browsers.

SELENIUM PROJECT

- Creating a Selenium Project
- **Selenium IDE** is a browser extension for **Chrome and Firefox** that allows **recording and playing back** Selenium test scripts without writing any code.
- It's useful for beginners and quick test automation.
- Step 1: Install Selenium IDE
- Step 2: Create a New Selenium Test Project
- Step 3: Record a Test
- Step 4: Edit and Verify Test Commands
- Step 5: Run the Test

FRONTEND TESTING

- Clone Existing React project: https://github.com/sonam-niit/frontend-testing-react.git
- Execute npm install
- Run test case: npm test