Course Material No. 7

SOFTWARE ENGINEERING 2

FE LARWA-HABLANIDA
Course Instructor

SOFTWARE TESTING FUNDAMENTALS 1

7

LEARNING OUTCOMES

At the end of the lesson, the learner will be able to:

- Explain the role and importance of software testing in the development process.
- Differentiate between various levels and types of software testing.
- Apply fundamental testing techniques to detect and report software defects.

RESOURCES NEEDED

For this lesson, you would need the following resources:

- PPT/Module
- Pencil and Paper

DISCUSSION:

INTRODUCTION TO SOFTWARE TESTING

Software Testing

Software Testing is the process of evaluating a software application or system to verify that it meets specified requirements and to identify defects.

- Testing is about finding defects, not proving perfection.
- Aims to ensure quality, reliability, and usability.
- Conducted throughout the Software Development Life Cycle (SDLC).

Importance of Software Testing

1. Ensures software meets user requirements.
2. Detects defects early, reducing cost and risk.
3. Builds customer confidence and trust.
4. Improves software performance and security.
5. Helps deliver quality software within time and budget.

Misconceptions About Software Testing

1. "Testing guarantees 100% bug-free software."
2. "Testing happens only after coding is finished."
3. "Anyone can test; it doesn't require skill."

- **Reality:** Testing improves quality but cannot prove perfection. It is a systematic, skilled process.

Verification vs. Validation

- **Verification**: "Are we building the product, right?" (Focuses on design, reviews, static testing)
- **Validation**: "Are we building the right product?" (Focuses on execution, meeting user needs)

Principles of Software Testing

1. Testing shows the presence of defects (not absence).
2. Exhaustive testing is impossible (you cannot test every input).
3. Early testing saves time and cost.
4. Defects cluster together (most defects are found in a few modules).

5. Pesticide paradox (repeating the same tests finds fewer defects; tests must evolve).
6. Testing is context-dependent.
7. Absence-of-errors fallacy (even if no bugs are found, software may still fail to meet user needs).

Example:

- *A bug in an e-commerce checkout system could result in financial losses and damage the company's reputation if not detected early.*

THE SOFTWARE TESTING LIFE CYCLE (STLC)

The **Software Testing Life Cycle (STLC)** is a sequence of specific activities conducted during the testing process to ensure that software quality objectives are met.

It defines what to do, when to do it, and who is responsible for it during the testing phases.

- **SDLC (Software Development Life Cycle):** Focuses on building software.
- **STLC (Software Testing Life Cycle):** Focuses on verifying and validating the software.

Benefits of STLC

1. Structured and systematic testing approach.
2. Early detection of defects.
3. Improved test coverage.
4. Better quality and reliable software.
5. Efficient resource and time utilization.

Phases of STLC

1. **Requirement Analysis**

   Requirement Analysis is the first phase of the Software Testing Life Cycle. It involves studying the software requirements (functional and non-functional) to identify what is testable, what needs clarification, and what risks may exist.

   - **Goal:** Understand what needs to be tested.

   - **Activities:**
     1. Analyze functional and non-functional requirements.
     2. Identify testable and non-testable requirements.
     3. Determine testing priorities and risks.

- **Deliverables:**

The **Requirement Traceability Matrix (RTM)** is a document that maps and traces requirements to corresponding test cases.

- To ensure all requirements are covered by test cases and no functionality is left untested.

Types of RTM

1. **Forward Traceability Matrix** – Maps requirements → Test Cases.
   - Ensures every requirement is tested.
2. **Backward Traceability Matrix** – Maps Test Cases → Requirements.
   - Ensures tests are linked to valid requirements (avoids unnecessary testing).
3. **Bi-directional Traceability Matrix** – Combines both forward and backward mapping.
   - Ensures 100% coverage and validates both directions.

Example:

Requirement ID	Requirement Description	Test Case ID(s)	Test Status
RQ-01	User must log in with a valid username & password	TC-01, TC-02, TC-03	Pass
RQ-02	Password must be at least 8 characters	TC-04	Fail
RQ-03	User can reset password via email link	TC-05, TC-06	Not Executed
RQ-04	The system must lock the account after 3 invalid login attempts	TC-07	Pass

2. **Test Planning**

Test Planning is the phase in STLC where the overall approach, objectives, resources, schedule, and scope of testing are defined.

- It answers: "*What to test, how to test, when*
- **Goal:** Define the testing strategy.

- **Activities:**

1. Decide scope, objectives, and testing approach.
2. Estimate effort, cost, and resources.
3. Identify risks and mitigation strategies.

- **Deliverables:** Test Plan document.

Test Plan Document Structure (IEEE 829 Standard), also called the Standard for Software Test Documentation. It is an internationally recognized standard that specifies the format and content of test documents used in software testing.

- It ensures consistency, clarity, and completeness across all testing activities.
- A **formal Test Plan** usually includes:

a. **Test Plan ID**
   - Unique ID or name for the test plan.
   - **Example:** *TP-OnlineBanking-Login-001*

b. **Introduction** – Overview of the project/system.
   - Purpose of the test plan.
   - Background about the system/application under test.
   - References (SRS, design docs, project plan).

c. **Test Items** – Features/modules to be tested.
   - Features/modules to be tested.
   - **Example:** *Login module, account balance module, fund transfer module.*

d. **Features to be Tested**
   - Specific requirements or functionalities to be verified.
   - **Example:** *Valid login, invalid login handling, password encryption.*

e. **Features Not to be Tested**
   - Out-of-scope features or excluded modules.
   - **Example:** *Mobile app login (if only web is being tested)*.

f. **Test Approach/Strategy**
   - **Testing Techniques:** Black-box, white-box, manual, automation.
   - **Levels of Testing**: Unit, integration, system, acceptance.
   - **Tools to be Used**: e.g., Selenium, JMeter.

g. **Test Environment Setup**
   - Conditions for determining whether a test item has passed or failed.
   - **Example:** *A test case passes if the actual output matches the expected result.*

h. **Entry and Exit Criteria**
   - Define when testing should be stopped or resumed.
   - **Example:** *Testing suspended if critical defects block execution.*

i. **Test Deliverables** (test cases, reports, defect logs)
   - Documents and outputs produced during testing.
   - **Example:** *Test cases, test scripts, defect reports, test summary report.*

j. **Responsibilities** (who will do what)
   - List of tasks and responsibilities.
   - **Example:** *Prepare test cases, set up test environment, execute tests, log defects.*

k. **Schedule & Milestones**
   - Timeline of test activities, milestones, deadlines.

l. **Risks and Contingency Plan**
   - Potential risks and mitigation strategies.

m. **Approvals** (sign-off by stakeholders)
   - Sign-offs from stakeholders (QA Lead, PM, Client).

➢ **Simplified Example Flow**

- **Test Plan** → defines how testing will be done.
- **Test Design & Test Cases** → prepare detailed scenarios to test requirements.
- **Test Procedure** → specifies execution order.
- **During Execution:** Testers record **Logs** and report **Incidents/Defects**.
- **End of Testing:** Prepare a **Summary Report** for stakeholders.

3. **Test Case Design / Test Development**

Test Case Design/Test Development is the process of creating test cases, test data, and test scripts based on requirements and test strategy. It ensures that all functionalities are verified and mapped to requirements (via RTM).

Test Case Design is the process of creating detailed test cases based on requirements, design documents, and user stories to ensure that every functionality is tested under positive and negative conditions.

- It converts requirements → test scenarios → test cases.
- **Goal:** Prepare test cases and data.
- **Deliverables:** Test Cases, Test Data, Test Scripts.

➢ **Activities:**

1. Write detailed test cases and test scripts.
2. Create test data sets.
3. Review and finalize test cases.

➢ **Test Development**

- It is also called Test Case Development.
- It is the process of:
  - Designing test scenarios and test cases.
  - Preparing test data.
  - Developing automation scripts (if automation is in scope).
- The goal is to ensure that every requirement has one or more test cases that verify it.

Examples

Test Case

- **Requirement ID:** RQ-01 (User must log in with valid credentials)

Field	Description
Test Case ID	TC-Login-001
Test Scenario	Verify login with a valid username & password
Pre-Condition	User account exists with username = "john123", password = "Test@123"
Test Steps	1. Navigate to the login page 2. Enter username "john123" 3. Enter password "Test@123" 4. Click Login
Test Data	Username: john123 Password: Test@123
Expected Result	User is successfully logged in and redirected to the Dashboard
Actual Result	(Filled during execution)
Status	(Pass/Fail)
Priority	High

Negative Test

- **Requirement ID**: RQ-02 (System must reject invalid login)

Field	Description
Test Case ID	TC-Login-002
Test Scenario	Verify login with invalid password
Pre-Condition	User account exists with username "john123"
Test Steps	1. Navigate to the login page 2. Enter username "john123" 3. Enter password "WrongPass" 4. Click Login
Test Data	Username: john123 Password: WrongPass
Expected Result	System displays error message: "Invalid username or password"
Actual Result	(Filled during execution)
Status	(Pass/Fail)
Priority	High

4. **Test Environment Setup**

Test Environment Setup is the process of creating the hardware, software, network, and configurations required to execute test cases.

- It is the platform where testing is performed (like a replica of production).
- **Goal:** Prepare the hardware/software environment.
- **Deliverables:** Test Environment Configuration document.

- **Activities:**
  1. Set up servers, databases, and network configurations.
  2. Install required tools.
  3. Validate the environment readiness.

Example

Test Environment for Online Banking Application

- **Hardware:**
  - *Windows Server 2019, 8 GB RAM, Intel Xeon*
- **Software:**
  - *Online Banking Application Build v1.0*
  - *Oracle Database 19c*
  - *Browsers: Chrome v110, Firefox v105*
- **Network:**
  - *Secure SSL certificates installed*
  - *VPN access for remote testers*
- **Test Data:**
  - *Dummy accounts: john123, mary456*
  - *Sample balances: $1,000, $5,000*
  - ***Tools:***
  - *JIRA (Defect Tracking)*
  - *Selenium WebDriver (Automation)*
  - *JMeter (Performance Testing)*

5. **Test Execution**

Test Execution is the phase where testers run the designed test cases in the prepared test environment using the specified test data.

The main goal is to validate that the software behaves as expected and to log defects when actual results differ from expected ones.

- **Goal:** Execute tests and record results.
- **Deliverables:** Test Execution Report, Defect Logs.
  - Executed Test Cases (with actual results).
  - Defect Reports (with severity and priority).
  - Test Execution Status Reports.
  - Updated Requirement Traceability Matrix (RTM).

- **Activities:**
  1. Run test cases and scripts.
  2. Record pass/fail results.
  3. Log defects in the defect tracking tool.

- **Test Execution Flow**

1. Test Case → Executed → Result?
    - **Pass** → Mark as *Passed*.
    - **Fail** → Log defect → Retest after fix.
    - **Blocked** → Mark as *Blocked* (due to environment/dependency issues).
    - **Not Run** → Mark if skipped.

Examples

Login Module (Online Banking App)

- **Test Case ID: TC-Login-001**
    - ***Steps:*** *Enter username john123, password Test@123, click Login.*
    - ***Expected Result:*** *User is redirected to Dashboard.*
    - ***Actual Result:*** *User redirected to Dashboard.*
    - ***Status:*** ☑ *Pass*
- **Test Case ID: TC-Login-002**
    - ***Steps:*** *Enter username john123, wrong password abc123, click Login.*
    - ***Expected Result:*** *Error message "Invalid username or password."*
    - ***Actual Result:*** *Application crashes.*
    - ***Status:*** ✗ *Fail*
    - ***Defect ID:*** *BUG-101 (Logged in JIRA).*
- **Test Case ID: TC-Login-003**
    - ***Steps:*** *Leave username blank, enter password, click Login.*
    - ***Expected Result:*** *Error "Username is required."*
    - ***Actual Result:*** *Error displayed correctly.*
    - ***Status:*** ☑ *Pass*

6. **Defect Reporting & Tracking**

- **Defect Reporting** is the process of documenting issues/bugs found during testing when the actual result differs from the expected result.
- **Defect Tracking** is the process of monitoring those defects from discovery → fixing → verification → closure.
- **Goal**: Ensure all defects are properly recorded, prioritized, fixed, retested, and closed.
- **Deliverables:** Updated Defect Reports.

- **Activities:**

1. Log new defects with details (steps, screenshots, severity).
2. Assign defects to developers.
3. Retest fixed defects and update status.

- **Defect Life Cycle (Bug Life Cycle)**

1. **New** – Tester reports a new defect.
2. **Assigned** – Defect assigned to a developer.

3. **Open/In Progress** – Developer works on fixing it.
4. **Fixed** – Developer marks as resolved.
5. **Retest** – Tester verifies the fix.
   - If successful → Closed.
   - If failed → Reopened.
6. **Deferred** – Fix postponed to future release.
7. **Rejected** – Considered not a defect / invalid.
8. **Closed** – Successfully fixed and verified.

- **Typical Defect Report Fields (IEEE 829/ISO Standard)**

1. **Defect ID** – Unique identifier.
2. **Title/Summary** – Short description of the issue.
3. **Description** – Detailed explanation with steps to reproduce.
4. **Steps to Reproduce** – Exact inputs, actions, and conditions.
5. **Expected Result** – What should happen.
6. **Actual Result** – What actually happened.
7. **Severity** – Impact on the system (Critical, Major, Minor, Trivial).
8. **Priority** – Order in which it should be fixed (High, Medium, Low).
9. **Module/Feature Affected** – Where the defect occurred.
10. **Environment** – OS, browser, database, build version.
11. **Attachments** – Screenshots, logs, videos.
12. **Reported By** – Tester's name.
13. **Assigned To** – Developer responsible.
14. **Status** – New, Assigned, Fixed, Retested, Reopened, Closed.
15. **Date Reported / Date Closed** – Tracking timely

Example

Defect Report (Online Banking – Login)

- **Defect ID:** BUG-101
- **Title:** Application crashes when an invalid password is entered.
- **Description:** The Login screen crashes when a user enters the wrong password and clicks "Login."
- **Steps to Reproduce:**
  1. Open the Login Page.
  2. Enter Username: *john123*.
  3. Enter Password: *WrongPass*.
  4. Click Login.
- **Expected Result:** Error message "Invalid username or password."

- **Actual Result:** Application crashes.
- **Severity:** Critical
- **Priority:** High
- **Module:** Login
- **Environment:** Chrome v110, Windows 10
- **Status:** New
- **Reported By:** QA Tester – Maria
- **Assigned To:** Dev – Alex

7. **Test Closure**

Test Closure is the phase where the testing team formally completes the testing process, evaluates the test cycle, and prepares documentation of results, metrics, and lessons learned.

- It's the stage where the team confirms that all planned testing activities are finished and delivers final reports to stakeholders.
- **Goal:** Formally close testing activities.
- **Deliverables:** Test Closure Report, Metrics & Analysis.

➢ **Activities:**

1. Evaluate test completion criteria.
2. Document lessons learned.
3. Archive test artifacts for future reference.

Example

Banking Application Test Closure Report (Summary)

Project: Online Banking System – Login & Funds Transfer Modules
Test Phase: System Testing
Duration: Aug 1 – Aug 30, 2025

Key Highlights:

- *Total Test Cases: 120*
- *Executed: 120 (100%)*
- *Passed: 110 (91.6%)*
- *Failed: 8 (6.6%)*
- *Blocked: 2 (1.8%)*
- *Critical Defects: 3 (all fixed & retested)*
- *Deferred Defects: 2 (minor UI issues, moved to next release)*

LEVELS OF SOFTWARE TESTING

Software testing is carried out at **different levels** to ensure complete validation of the system. Each level focuses on a specific aspect of the software.

1. **Unit Testing**

Unit Testing is the process of testing the **smallest testable part of an application** (called a *unit*) in isolation to ensure it works correctly. Testing individual **modules/components** of the software in isolation.

- A **unit** can be a function, method, procedure, or class in a program.

➢ **When to Perform**

- During the coding phase of SDLC.
- Before integration testing.
- Each time new code is added or modified.
- **Performed By:** Developers (sometimes with testers in TDD/Agile).
- **Objective:** Verify that each unit of code works as expected.
- **Tools:** JUnit, NUnit, PyTest, xUnit.

Example:

- *Testing the login() function to confirm it correctly validates username/password.*

2. **Integration Testing**

Integration Testing is the process of testing the interaction between two or more integrated modules/components to verify that they work together as expected.

Even if modules work correctly in isolation (via Unit Testing), they may fail when combined due to interface issues, data flow problems, or mismatched assumptions.

- Testing the interaction between integrated modules to ensure they work together correctly.
- **Objective:** Verify **data flow and communication** between components.
- **Performed By:** Developers & Testers.

➢ **Approaches:**

a. **Big Bang Integration** – Integrate all modules at once.

- Combine all modules and test at once.
- Simple, but defects are harder to isolate.

b. **Incremental Integration** – Add and test modules step by step.

- Add and test modules step by step.
  - **Top-Down Approach:** Start from top-level modules, integrate downwards.
  - **Bottom-Up Approach:** Start from low-level modules, integrate upwards.
  - **Sandwich/Hybrid Approach:** Combines Top-Down + Bottom-Up.

Example:

Banking Application (Login + Account Balance)

- **Modules:**
  - **Module A***: Login Service*
  - **Module B***: Account Balance Service*
- **Test Scenario:**
  - *User logs in with valid credentials.*
  - *System retrieves and displays the correct account balance.*
- **Possible Defects Detected in Integration Testing:**
  - *Login works fine, but the user ID is not passed correctly to the account module.*
  - *Database connection mismatch.*
  - *Incorrect error handling between services.*

3. **System Testing**

System Testing is the process of testing the entire software system as a whole in a controlled environment to ensure it meets specified functional and non-functional requirements.

- It validates the system's end-to-end behavior before acceptance testing.
- Testing the entire system as a whole in a test environment that closely resembles production.
- Validate that the system meets functional and non-functional requirements.
- **Performed By:** Independent Testing Team (QA).

➢ **Types:**

a. Functional Testing (end-to-end scenarios).

b. Non-functional Testing (performance, security, usability, etc.).

Example

Online Banking System

- **Scenario:** *Verify funds transfer between two accounts.*
- **Pre-condition:**
  - *User1 has $1,000 in Account A.*
  - *User2 has $500 in Account B.*
- **Steps:**
  1. *Login as User1.*
  2. *Navigate to Funds Transfer.*
  3. *Enter amount = $200, destination account = User2.*
  4. *Submit transaction.*
- **Expected Result:**
  - User1's balance = $800.
  - User2's balance = $700.
  - Transaction history updated.
- **Actual Result:** *To be filled after execution.*
- **Status:** Pass / Fail

4. **Acceptance Testing**

Acceptance Testing is the process of verifying whether a software system meets the **business requirements and user expectations** and is ready for delivery to production.

- It is the last phase of testing in the Software Testing Life Cycle (STLC).
- Testing the system against business requirements to determine if it is ready for release.
- **Performed By:** End-users, clients, or QA team.
- **Objective:** Gain **final approval** before deployment.

➢ **Types:**

a. **User Acceptance Testing (UAT):** Validates functionality for end-users.
   - Performed by end-users to ensure the system meets business needs.
   - **Example:** HR team testing payroll system.

b. **Business Acceptance Testing (BAT):** Ensures business rules are met.
   - Conducted by business representatives to validate business rules.

c. **Contract & Regulation Acceptance Testing:** Checks compliance.
   - Ensures software meets contractual obligations.
   - Ensuring system complies with legal and regulatory standards (e.g., GDPR, HIPAA).

d. **Alpha Testing:** Conducted internally by QA/end-users before release.
   - Done in development environment by internal staff.

e. **Beta Testing:** Conducted by a limited set of external users in a real environment.
   - Done in the real world by selected external users.

Example

E-commerce Website (UAT Scenario)

- **Requirement:** A customer should be able to place an order successfully.
- **Pre-condition:**
  - *Customer is registered and logged in.*
  - *Product is in stock.*
- **Steps:**
  1. *Search for a product.*
  2. *Add it to the cart.*
  3. *Proceed to checkout.*
  4. *Enter shipping and payment details.*
  5. *Confirm the order.*
- **Expected Result:**
  - *Order confirmation page is displayed.*

  - *Order ID is generated.*
  - *Customer receives confirmation email.*
- **Status:** Pass / Fail

TYPES OF SOFTWARE TESTING

Software testing can be classified into different types based on **scope, method, objective, and execution style.**

1. **Based on Knowledge of Code**

Testing types here are classified according to how much the tester knows about the internal code and structure of the software.

a. **White Box Testing**

- Tester has full knowledge of the internal code, logic, and structure.
- Also called **Glass Box, Structural, or Clear Box Testing.**
- Performed mainly by **developers**.
- **Focus**: Code logic, loops, conditions, data flow.

Example:

- *Checking if a loop executes the correct number of times, or if all branches in an if-else statement are covered.*

b. **Black Box Testing**

- Testing where the tester does not know the internal code. They only focus on inputs and outputs.
- Tester only knows inputs and expected outputs, no knowledge of code.
- Based on **requirements and functionality**.
- Performed mainly by **testers/QA engineers**.
- **Focus**: Does the system give correct output for a given input?

Example:

- *Entering a valid username and password to check if the login is successful, without knowing how the authentication is coded.*

c. **Gray Box Testing**

- Testing where the tester has **partial knowledge** of the internal structure, along with functional knowledge.
- Partial knowledge of the system.
- Combines **White Box + Black Box**.
- Performed by **testers or developers**.
- **Focus**: Testing functional flow + internal design aspects**.**

Example:

- *Testing a web application where the tester knows the database schema but tests the functionality through the user interface (UI).*

2. **Based on Scope (Levels of Testing)**

Types of Software Testing Based on Scope, which refers to the level at which testing is performed in the software.

a. **Unit Testing**

- Smallest piece of code (functions, methods).
- Testing types here are classified according to how much the tester knows about the internal code and structure of the software.
- Performed by **developers** (sometimes with testers).
- Focus on the **correctness of code logic**.
- Usually **automated**.

Example:

- *Testing a function that calculates the discount percentage in an e-commerce app.*

b. **Integration Testing**

- Interaction between modules.
- Testing the interaction between integrated modules to ensure they work together.
- Done after unit testing.
- Focus on data flow and module communication.

Example:

- *Testing the interaction between the login module and the database.*

c. **System Testing**

- Complete system validation.
- Testing the entire system as a whole in a controlled environment.
- Performed by an **independent QA team**.
- Focus on **end-to-end functionality and non-functional requirements.**
- Includes **functional + non-functional testing.**

Example:

- *Testing an online banking system: login → transfer funds → logout, to ensure the whole process works correctly*

d. **Acceptance Testing**

- End-user validation.

- Testing is performed to validate whether the system is **ready for delivery** and meets **business/user requirements.**
- Performed by **end-users, clients, or business stakeholders.**
- Final stage before deployment.

Example:

- *Client tests an e-commerce website to ensure that users can search, order, and pay successfully before go-live.*

3. **Based on Functionality**

a. **Functional Testing**

- Testing that verifies whether the software **functions correctly** according to the requirements
- Ensures software performs required functions as per SRS/BRD.
- Black-box approach (focuses on inputs & outputs, not code).
- Ensures each feature works as intended.
- Checks business logic, workflows, and data processing.
- **Includes:** Smoke Testing, Sanity Testing, Regression Testing, UAT

Example:

In an e-commerce app, testing that:

- *A user can add items to a cart.*
- *Checkout works correctly with different payment methods.*

b. **Non-Functional Testing**

- Testing that checks the quality attributes of the software rather than specific behaviors or features.
- Ensures the software is usable, secure, scalable, and reliable.
- Often requires specialized tools.
- Includes performance, security, and usability testing.
- Tests quality attributes (performance, usability, security, etc.).
- **Includes:** Performance Testing, Load Testing, Stress Testing, Security Testing, Usability Testing, and Reliability Testing.

Example

In the same e-commerce app, testing that:

- *The site can handle 1,000 users checking out simultaneously (Performance).*
- *Credit card information is encrypted and secure (Security).*
- *The site works on Chrome, Safari, and mobile browsers (Compatibility).*
- *Pages load within 2 seconds (Performance).*

4. **Based on Execution**

Execution refers to how test cases are carried out – whether by humans or with the help of automation tools.

a. **Manual Testing**

- Tester executes test cases step by step.
- Best for exploratory, usability, and ad-hoc testing.
- Detects issues related to user experience (UX) that automation might miss.
- Test cases executed manually by testers.
- Best for **exploratory, usability, ad-hoc testing.**

Example:

- *A tester manually navigates through an e-commerce checkout process to verify if adding to cart, applying a coupon, and completing payment work correctly.*

b. **Automation Testing**

- Testing where scripts or tools automatically execute test cases, compare actual vs expected results, and report defects.
- Test cases executed using tools/scripts.
- Best for **repetitive and regression testing.**
- Best for repetitive, regression, and performance testing.
- Requires initial investment in tools, frameworks, and scripts.
- Provides fast execution and reusability.
- **Tools**: Selenium, JUnit, PyTest, Cypress, JMeter.

Example

E-Commerce Website Checkout

- **Functional Testing**: *Can the user add an item to the cart and check out?*
- **Regression Testing**: *After updating the payment gateway, does checkout still work?*
- **Performance Testing**: *Can the system handle 1,000 users checking out at once?*
- **Security Testing**: *Is payment data encrypted?*
- **Usability Testing**: *Is the checkout process simple for new users?*

REFERENCES

- Pressman, Roger S, Software Engineering: A Practitioner's Approach, 9th Edition, Published by Mc Graw-Hill (2019)
- Sommerville, Ian, Software Engineering 10th Edition, Published by Pearson Education, Inc (2016)
- Bass, Len, Clements, Paul, & Kazman, Rick., Software Architecture in Practice (3rd Edition). Addison-Wesley, 2012.