Chapter 1: Introduction to Software Testing

1.1 What is Software Testing?

Software testing is the process of **evaluating software** to ensure it meets specified requirements and works as intended. It involves executing a program or system to identify **defects, gaps, or missing requirements** compared to the actual expected output.

- **Goal of Testing**: Deliver a **high-quality product** by ensuring reliability, performance, and user satisfaction.
- **Key Point**: Testing does not guarantee a bug-free product; instead, it reduces the risk of failure.

1.2 Why is Testing Important?

1. **Quality Assurance** – Ensures that the product meets the required standards.
2. **Cost Effectiveness** – Early detection of defects reduces fixing costs.
3. **Customer Satisfaction** – Provides a reliable, user-friendly product.
4. **Prevention of Failures** – Detects critical errors before deployment.

💡 *Mnemonic*: **QCCP** → *Quality, Cost, Customer, Prevention*.

1.3 Objectives of Software Testing

- Detect defects and errors.
- Ensure the product meets **functional** and **non-functional** requirements.
- Improve product reliability and performance.
- Provide confidence to stakeholders about the product.

1.4 Principles of Software Testing

1. **Testing shows presence of defects** (not absence).
2. **Exhaustive testing is impossible** – you can't test everything.
3. **Early testing saves time and money**.

4. **Defects cluster together** – most bugs lie in a few modules.
5. **Pesticide paradox** – repeating the same tests won't find new bugs.
6. **Testing is context dependent** – strategy depends on the system.
7. **Absence-of-errors fallacy** – no errors ≠ useful product.

💡 *Mnemonic*: **PEEDPAC** → *Presence, Exhaustive, Early, Defect clusters, Pesticide, Context, Absence fallacy*.

1.5 Types of Testing

- **Manual Testing**: Performed by humans without automation tools.
- **Automation Testing**: Performed using tools/scripts to speed up execution.

✅ **Summary**

- Testing improves quality and reduces risk.
- It has principles, objectives, and methods.
- Both manual and automation testing are essential in industry.

Chapter 2: Software Development Life Cycle (SDLC)

2.1 What is SDLC?

The **Software Development Life Cycle (SDLC)** is a structured process followed to build high-quality software. It defines **phases, tasks, and deliverables** in software development.

2.2 Phases of SDLC

1. **Requirement Analysis**
    - Collect requirements from clients and stakeholders.
    - Document Functional & Non-Functional Requirements (FRS, SRS).
2. **Feasibility Study**
    - Analyze cost, time, and technology feasibility.
    - Conduct **Risk Analysis**.
3. **System Design**
    - High-Level Design (HLD): architecture and modules.
    - Low-Level Design (LLD): detailed internal design.
4. **Implementation (Coding)**
    - Actual coding begins.
    - Programming languages & frameworks are used.
5. **Testing**
    - QA team verifies functionality.
    - Unit testing, integration testing, system testing, acceptance testing.
6. **Deployment**
    - Software is released to production.
    - Can be staged (Beta, Pilot, Production).
7. **Maintenance**
    - Fix post-release issues.
    - Provide updates and enhancements.

💡 *Mnemonic*: **RFD-TDM** → *Requirements, Feasibility, Design, Testing, Deployment, Maintenance*.

2.3 SDLC Models

1. **Waterfall Model** – Sequential, rigid structure.
2. **V-Model** – Validation & verification go in parallel.
3. **Incremental Model** – Develop in increments/parts.
4. **Agile Model** – Iterative, adaptive, customer-focused.
5. **Spiral Model** – Risk-based iterative development.

2.4 Importance of SDLC

- Provides a **systematic approach** to development.
- Ensures **quality, cost, and time efficiency**.
- Helps in **risk management**.

✅ **Summary**

- SDLC is a step-by-step framework for software development.
- Consists of **7 major phases**.
- Different models exist (Waterfall, V-Model, Agile, etc.), chosen based on project needs.

Chapter 3: Software Development Life Cycle (SDLC) Models

3.1 Introduction to SDLC

- **Definition**:
  The **Software Development Life Cycle (SDLC)** is a systematic process of planning, creating, testing, and deploying software.
- **Purpose**:
  - Ensures software meets **customer requirements**.
  - Provides a **structured framework** for quality, cost, and time efficiency.
  - Reduces risks of project failure.

3.2 Major SDLC Models

Different SDLC models are used depending on project needs. Each has **advantages, disadvantages, and suitable scenarios**.

3.2.1 Waterfall Model

- **Definition**: A **linear, sequential model** where each phase must be completed before the next begins.
- **Phases**:
  1. Requirements
  2. Design
  3. Implementation (Coding)
  4. Testing
  5. Deployment
  6. Maintenance

Diagram (Textual Representation):

Requirements → Design → Implementation → Testing → Deployment → Maintenance

- **Advantages**:
  ✅ Simple and easy to use.

✅ Well-suited for projects with **clear, fixed requirements**.
✅ Each phase has **specific deliverables**.

- **Disadvantages**:
  ❌ Rigid – changes are costly.
  ❌ No working software until late stages.
  ❌ Poor model for complex or uncertain projects.
- **Best suited for**: Government, defense, or projects with **well-defined requirements**.

3.2.2 Spiral Model

- **Definition**: A **risk-driven model** combining iterative development with risk analysis.
- **Process**: Each cycle (spiral) includes:
  1. Planning
  2. Risk Analysis
  3. Engineering (Development + Testing)
  4. Evaluation

Diagram (Textual Representation):

(Planning)

↑

Evaluation ← Risk Analysis

↓

Engineering

- **Advantages**:
  ✅ Focuses on **risk management**.
  ✅ Allows **customer feedback** at each iteration.
  ✅ Better suited for **large, high-risk projects**.
- **Disadvantages**:
  ❌ Expensive compared to Waterfall.
  ❌ Requires skilled risk analysis team.

- **Best suited for**: Large projects with **uncertainty** and **high risk** (e.g., banking systems).

3.2.3 V-Model (Verification & Validation Model)

- **Definition**: Extension of Waterfall, where **each development phase has a corresponding testing phase**.
- **Process**:
    - Requirements ↔ Acceptance Testing
    - Design ↔ System Testing
    - Detailed Design ↔ Integration Testing
    - Coding ↔ Unit Testing

Diagram (Textual Representation):

Requirements ↔ Acceptance Test

Design ↔ System Test

Detailed Design ↔ Integration Test

Coding ↔ Unit Test

- **Advantages**:
  ✅ Testing activities are planned early.
  ✅ Higher chance of detecting defects early.
  ✅ Clear verification and validation process.
- **Disadvantages**:
  ❌ Costly and rigid.
  ❌ Not good for projects with frequently changing requirements.
- **Best suited for**: Safety-critical projects (e.g., medical, aerospace).

3.2.4 Agile Model (Overview)

- **Definition**: An **iterative, incremental approach** that delivers working software in **small cycles (sprints)**.
- **Principles (Agile Manifesto)**:
    - Individuals & interactions over processes.

    - Working software over documentation.
    - Customer collaboration over contracts.
    - Responding to change over rigid plans.
- **Advantages**:
  ✅ Flexibility – requirements can change.
  ✅ Regular customer feedback.
  ✅ Faster delivery of usable product.
- **Disadvantages**:
  ❌ Requires experienced team.
  ❌ Difficult for very large projects without proper planning.
- **Best suited for**: Projects where **requirements evolve rapidly** (e.g., startups, web apps).

3.3 Comparison of SDLC Models

Model	Flexibility	Cost	Risk Handling	Best For
Waterfall	Low	Low	Poor	Fixed requirements
Spiral	High	High	Excellent	Risky, large projects
V-Model	Medium	Medium	Good	Safety-critical
Agile	Very High	Medium	Good	Dynamic requirements

3.4 Interview-Focused Q&A

Q1. Difference between Waterfall and Agile?

- Waterfall: Sequential, rigid, no customer involvement after start.
- Agile: Iterative, flexible, customer involved throughout.

Q2. Why is Spiral model considered risk-driven?

- Because every iteration includes **risk analysis and mitigation**.

Q3. What is the main drawback of the V-Model?

- It is rigid and not suitable for **changing requirements**.

Q4. Which SDLC model is best for safety-critical projects?

- **V-Model**.

Q5. Which SDLC model is best when requirements are unclear?

- **Agile** or **Spiral**.

🎯 Memory Aid (Mnemonic)

To remember the four main SDLC models:
"Wise Students Value Agility"

- **W** → Waterfall
- **S** → Spiral
- **V** → V-Model
- **A** → Agile

Chapter 4: Verification and Validation (V&V) Model

4.1 Introduction

Software quality depends not only on *what we build* but also on *how we check it*. The **Verification and Validation (V&V) Model** ensures that software is built correctly and also meets user needs.

- **Verification** = "Are we building the product right?"
- **Validation** = "Are we building the right product?"

This model is often represented by the **V-Model** in SDLC.

4.2 Difference Between Verification and Validation

Aspect	Verification	Validation
Definition	Process of evaluating work-products (documents, design, code) without executing the software	Process of evaluating the final product by executing it
Focus	Ensures product is built as per requirements and design	Ensures product meets business needs
Techniques	Reviews, walkthroughs, inspections, static analysis	Testing (functional, system, UAT)
Performed by	Developers, QA team	QA team, end-users, stakeholders
When	During development (early phase)	After build, before release

---

4.3 The V-Model of Software Development

The **V-Model** is an extension of the Waterfall Model. Every development phase has a corresponding testing phase.

Requirements → Acceptance Testing

System Design → System Testing

Architecture Design → Integration Testing

Module Design → Unit Testing

Left side of V = Development

- Requirement Analysis
- System Design
- Architecture Design
- Module Design

Right side of V = Testing

- Acceptance Testing (validates requirements)
- System Testing (validates system design)
- Integration Testing (validates architecture)
- Unit Testing (validates modules)

4.4 Advantages of V&V Model

- Defects are detected early → saves cost.
- Each phase is validated → ensures correctness.
- Clear relationship between dev phases and testing phases.
- High reliability for critical systems (banking, healthcare).

4.5 Disadvantages of V&V Model

- Rigid, not flexible for changes (similar to Waterfall).
- Requires good documentation.
- Costly for small projects.

4.6 Example Case

Suppose we are building a **Banking App**:

- **Verification:** Check whether login screen design follows the requirement specs.
- **Validation:** Test whether real users can log in and perform transactions successfully.

4.7 Interview Questions

1. **Q:** What is the main difference between Verification and Validation?
   A: Verification checks *process* (reviews, static checks) while Validation checks *product* (testing with execution).
2. **Q:** Why is the V-Model called a Verification and Validation model?
   A: Because every development stage (verification) has a corresponding testing stage (validation).
3. **Q:** In which phase can we start verification activities?
   A: As early as requirement analysis.
4. **Q:** Can validation exist without verification?
   A: No, if requirements/design are not verified, testing may miss defects.

✅ Quick Recap Mnemonic

- **V = Verify (left) + Validate (right)**
- **Veri = Docs, Vali = Users**
- **Each step tested with its mirror step**

Chapter 5: Software Testing Documentation

5.1 Introduction

Software Testing Documentation is the **set of artifacts and records created during the testing process**.
It provides a structured way to **plan, execute, track, and report testing activities**.
Documentation acts as **proof of quality** and ensures **traceability** of requirements to test cases.

💡 *Think of documentation as the "map" for testers → it tells what to test, how to test, and what was found.*

5.2 Importance of Testing Documentation

- ✅ Ensures **clarity** of testing scope and objectives.
- ✅ Helps in **communication** between testers, developers, and stakeholders.
- ✅ Provides **traceability** of requirements to test cases.
- ✅ Serves as **evidence** for audits and compliance.
- ✅ Acts as a **reference** for future testing cycles or regression.

Mnemonic → *"PCTER"*

- **P**roof of work
- **C**larity of scope
- **T**raceability
- **E**vidence for audit
- **R**eference for future

5.3 Key Testing Documents

1. Test Plan

- A **high-level document** describing:
    - Scope of testing
    - Testing objectives

    - Resources required
    - Schedules and milestones
    - Risks and mitigation strategies
- Example: *"We will test the login module, exclude third-party integrations, and finish by 10th Sept."*

---

2. Test Case

- A **detailed set of conditions** and steps to validate functionality.
- Components:
    - Test case ID
    - Preconditions
    - Steps
    - Expected Result
    - Actual Result
    - Status (Pass/Fail)

📌 *Example:*

- **TC-101**: Verify login with valid credentials
    - Precondition: User is registered
    - Steps: Enter username, password → Click login
    - Expected: Redirect to dashboard

---

3. Test Scenario

- High-level description of **what to test**, without detailed steps.
- Example: *"Check login functionality with valid and invalid inputs."*

---

4. Requirement Traceability Matrix (RTM)

- A table mapping **requirements to test cases**.
- Ensures **no requirement is left untested**.

📌 Example:

Requirement ID	Requirement Description	Test Case ID(s)
RQ-001	Login with valid credentials	TC-101, TC-102
RQ-002	Password reset functionality	TC-103, TC-104

---

5. Test Report

- Document prepared after test execution.
- Contains:
    - Test summary
    - Passed/failed cases
    - Defects found
    - Risks remaining

5.4 Types of Documentation in SDLC Context

- **Before Testing** → Test Plan, Test Scenarios
- **During Testing** → Test Cases, RTM
- **After Testing** → Test Reports, Defect Reports

5.5 Interview Questions

Q1. Why is documentation important in testing?

- Proves testing activities
- Ensures traceability
- Acts as future reference

Q2. Difference between Test Case and Test Scenario?

- Test Case = Detailed step-by-step execution
- Test Scenario = Broad functionality to be tested

Q3. What is RTM and why is it used?

- Requirement Traceability Matrix links requirements → test cases → defects
- Ensures full coverage

Q4. What are the components of a Test Plan?

- Scope, objectives, schedule, resources, risks, deliverables

---

5.6 Summary

- Documentation = backbone of testing.
- Key docs: **Test Plan, Test Cases, Test Scenarios, RTM, Test Reports**.
- Provides **clarity, traceability, and evidence** for testing activities.

Chapter 6: Defect / Bug Life Cycle – Its Importance

6.1 What is a Defect/Bug?

- A **Defect (Bug)** is any flaw in the software that causes it to behave differently from the expected result.
- If a tester finds that the actual output is not matching the expected output → it is reported as a defect.
- In industry, terms like *defect*, *bug*, *issue*, and *incident* are often used interchangeably.

Example:

- Expected: Login with correct credentials should redirect to the dashboard.
- Actual: Login with correct credentials shows an error → Bug reported.

6.2 Defect Life Cycle (Bug Life Cycle)

The **Defect Life Cycle** is the journey of a defect from **identification** to **closure**.

🔄 **Stages of Defect Life Cycle:**

1. **New** → Tester logs a new defect.
2. **Assigned** → Project manager/test lead assigns defect to a developer.
3. **Open** → Developer starts analyzing and fixing the defect.
4. **Fixed/Resolved** → Developer makes necessary code changes and marks as fixed.
5. **Retest** → Tester retests the functionality.
    - If **pass** → defect marked as **Verified/Closed**.
    - If **fail** → defect marked as **Reopened** and sent back to developer.
6. **Deferred** → If defect is minor or low priority, it may be postponed.
7. **Rejected** → If developer disagrees that it is a defect.
8. **Duplicate** → If defect is same as an already reported one.
9. **Closed** → Once tester verifies the fix successfully.

6.3 Defect Life Cycle Diagram

[New] → [Assigned] → [Open] → [Fixed/Resolved]

↓ ↑

[Rejected] [Deferred] [Reopened]

↓

[Retest] → [Closed]

6.4 Importance of Defect Life Cycle

- Ensures **standardized process** for defect handling.
- Helps in **tracking progress** of defect resolution.
- Provides **clarity** among testers and developers.
- Assists in **measuring quality** of the product.
- Supports **project management decisions** (e.g., go/no-go release).

6.5 Example – Defect Life Cycle in Action

1. Tester reports: *"Logout button not working in Chrome."* → **New**
2. Test Lead assigns to Dev → **Assigned**
3. Developer accepts and investigates → **Open**
4. Fix applied → **Fixed**
5. Tester retests → Still not working → **Reopened**
6. Developer fixes again → Retest successful → **Closed**

6.6 Interview Questions

Q1. What is a defect?
A defect is a deviation from expected behavior of the application, found during testing.

Q2. What are the different statuses of a defect?
New, Assigned, Open, Fixed, Retest, Reopened, Deferred, Rejected, Duplicate, Closed.

Q3. What is the difference between 'Defect', 'Bug', and 'Error'?

- **Error** → mistake made by developer.

- **Bug/Defect** → found during testing when expected ≠ actual.
- **Failure** → when bug reaches production and causes malfunction.

Q4. Why is defect life cycle important?
It standardizes defect handling, improves collaboration, and ensures product quality.

Chapter 7: What is Manual Testing?

7.1 Introduction

Manual Testing is the process of **executing test cases manually** without using any automation tools. A tester plays the role of an end-user, checking whether the developed software works as expected and identifying defects.

📌 **Definition**:
Manual Testing is a type of software testing where test cases are executed by a human without using automation tools to ensure the software behaves as per requirements.

7.2 Key Characteristics of Manual Testing

- **Human-centric** → Conducted by testers, not scripts.
- **Exploratory in nature** → Allows testers to discover unexpected issues.
- **Best for UI/UX** → Humans can judge usability better than machines.
- **Low cost initially** → No expensive automation setup needed.
- **Time-consuming** → Repeated execution of test cases is slower.

7.3 Objectives of Manual Testing

1. Verify if the application meets business requirements.
2. Detect software defects before release.
3. Ensure usability and user experience.
4. Validate performance under different environments.

7.4 Types of Manual Testing

Manual testing covers multiple subtypes, often performed before automation is introduced:

1. **Black Box Testing** – Focus on inputs/outputs without internal code knowledge.
2. **White Box Testing** – Tester verifies logic, loops, and paths inside the code.

3. **Grey Box Testing** – Partial knowledge of internal working with functional tests.
4. **Exploratory Testing** – Testing without predefined test cases, relies on tester experience.
5. **Ad-hoc Testing** – Informal, random testing to find defects quickly.

7.5 Process of Manual Testing

The process follows the **STLC (Software Testing Life Cycle):**

1. **Requirement Analysis** → Understand what needs to be tested.
2. **Test Planning** → Define strategy, scope, tools (if any), risks.
3. **Test Case Development** → Write detailed test cases.
4. **Test Environment Setup** → Prepare hardware/software for testing.
5. **Test Execution** → Execute test cases manually.
6. **Defect Reporting** → Log any found defects into a defect management system.
7. **Test Closure** → Analyze results and document lessons learned.

7.6 Advantages of Manual Testing

✓ Best for **UI/UX testing** (human judgment needed).
✓ Useful for **short-term projects**.
✓ Detects unexpected issues via exploratory/ad-hoc testing.
✓ Easy to start → No automation knowledge needed.

7.7 Disadvantages of Manual Testing

❌ Time-consuming for large projects.
❌ Prone to **human error**.
❌ Regression testing becomes inefficient.
❌ Cannot simulate **performance/load testing**.

7.8 When to Prefer Manual Testing?

- For **new applications** in early stages.
- When UI/UX feedback is essential.

- When requirements change frequently.
- For **exploratory or ad-hoc testing**.
- For **small projects** with limited budget and time.

7.9 Example

Imagine you are testing a **Login Page** manually:

- Enter correct username & password → Expect success.
- Enter wrong username → Expect error message.
- Leave fields blank → Expect validation warning.

Here, the tester plays the **end-user role** and manually verifies outputs.

7.10 Common Interview Questions

1. What is Manual Testing?
2. Why is Manual Testing important even with Automation tools available?
3. What are the types of Manual Testing?
4. Explain the process you follow while doing Manual Testing.
5. What are the advantages & disadvantages of Manual Testing?
6. Give an example where Manual Testing is preferred over Automation.
7. Difference between Exploratory and Ad-hoc Testing.

7.11 Quick Mnemonic

👉 **"H-E-P-T-A"** for remembering **Manual Testing Process**

- **H** – Highlight Requirements (Requirement Analysis)
- **E** – Establish Plan (Test Planning)
- **P** – Prepare Cases (Test Case Development)
- **T** – Test Execution
- **A** – Analyse & Report (Defects + Closure)

Chapter 8: Automation Testing – What & Why

8.1 Introduction to Automation Testing

- **Automation Testing** is the process of using software tools/scripts to perform tests on applications automatically.
- Unlike manual testing, where a human executes test cases, automation uses predefined scripts, tools, and frameworks.

👉 **Key Idea**: Automation = Faster, repeatable, reliable testing.

8.2 Why Automation Testing?

Automation is **not a replacement** for manual testing but a **complement**.

Benefits of Automation:

1. **Speed** – Faster execution of test cases compared to manual effort.
2. **Reusability** – Scripts can be reused across builds/releases.
3. **Coverage** – Large test suites can be executed quickly.
4. **Reliability** – No human error in repetitive tasks.
5. **CI/CD Integration** – Works seamlessly in DevOps pipelines.
6. **Cost-effective (long term)** – Though initial investment is high, it saves cost over time.

8.3 When to Automate?

Not all tests should be automated. Automation is best for:

- **Regression Testing** → Running same tests repeatedly.
- **Smoke & Sanity Testing** → Quick validation of builds.
- **Performance/Load Testing** → Requires thousands of users (JMeter, LoadRunner).
- **Cross-browser/Device Testing** → Selenium Grid, Appium.
- **Data-driven Testing** → Where same test is run with different inputs.

⚠️ **Don't Automate**:

- New functionalities (need manual exploratory testing first).
- One-time test cases.
- Usability and UI look & feel.

8.4 Popular Automation Tools

- **Selenium** – Web automation (most popular).
- **JUnit/TestNG** – Unit & regression testing frameworks.
- **Appium** – Mobile testing automation.
- **JMeter** – Performance testing.
- **Cypress, Playwright** – Modern end-to-end testing tools.

8.5 Automation Testing Process

1. **Test Tool Selection** – Choose the right tool (Selenium, JMeter, etc.).
2. **Define Scope of Automation** – Select modules/test cases to automate.
3. **Test Planning** – Decide framework, test data, environment setup.
4. **Test Script Development** – Write automation scripts.
5. **Test Execution** – Run scripts on builds (CI/CD).
6. **Maintenance** – Update scripts when application changes.

8.6 Manual vs Automation Testing

Feature	**Manual Testing**	**Automation Testing**
Speed	Slow	Fast
Cost (initial)	Low	High
Long-term cost	High	Low
Best for	Exploratory, usability	Regression, performance
Human error	Possible	Eliminated

Feature	Manual Testing	Automation Testing
Reusability	No	Yes

8.7 Advantages & Disadvantages

✅ **Advantages**:

- Faster regression cycles.
- Better accuracy.
- Increases coverage.

❌ **Disadvantages**:

- High initial setup cost.
- Not suitable for UI/UX.
- Requires skilled testers.

8.8 Interview Questions (Automation Basics)

Q1. What is automation testing?
👉 Automation testing is using tools/scripts to execute test cases automatically instead of manual execution.

Q2. Why do we need automation testing?
👉 To reduce repetitive manual effort, speed up regression, and improve test reliability.

Q3. Can automation testing completely replace manual testing?
👉 No, manual testing is still required for exploratory, usability, and UI validation.

Q4. Which tests should be automated?
👉 Regression, smoke, sanity, performance, cross-browser, and repetitive tests.

Q5. Name some automation testing tools you know.
👉 Selenium, Appium, JMeter, Cypress, Playwright.

📌 **Mnemonic to Remember Benefits of Automation (SPEED)**

- **S** → Saves Time
- **P** → Prevents Human Error

- **E** → Enhances Coverage
- **E** → Ensures Reusability
- **D** → DevOps Friendly

Chapter 9: Test Automation Process

Automation testing is more than just writing scripts; it's a **structured process** that ensures testing is reliable, repeatable, and adds value to the software development lifecycle.

🔹 9.1 What is Test Automation Process?

The **Test Automation Process** refers to the **methodology of planning, designing, developing, executing, and maintaining automated test scripts**.
It ensures that repetitive, time-consuming manual test cases are executed faster and with high accuracy.

Objective:

- Increase test efficiency
- Improve test coverage
- Reduce human error
- Save time in regression testing

🔹 9.2 Stages of Test Automation Process

1. Feasibility Analysis

- Identify which test cases can and should be automated.
  ✅ Good candidates: Regression tests, smoke tests, repetitive tasks
  ❌ Bad candidates: One-time tests, UI tests with frequent changes

2. Tool Selection

- Choose the right automation tool based on:
  - Application type (web, mobile, desktop, API)
  - Budget (open-source vs. licensed tools)
  - Team expertise
  - Integration needs (CI/CD pipelines)

👉 Popular tools: Selenium, Playwright, Cypress, JUnit, PyTest, JMeter, Postman

3. Test Planning

- Define scope of automation
- Identify resources & skill requirements
- Estimate timelines & cost
- Decide reporting & logging mechanism

4. Test Case Design

- Convert manual test cases into automation scripts
- Use **modular, reusable functions**
- Follow **naming conventions** and **data-driven testing** principles

5. Test Environment Setup

- Configure the required environment
- Install dependencies, drivers, and plugins
- Ensure test data availability
- Integrate with CI/CD (Jenkins, GitHub Actions, GitLab CI)

6. Test Script Development

- Write automation scripts using chosen framework
- Follow **coding best practices** (reusability, maintainability)
- Use page object model (POM) for UI testing

7. Test Execution

- Run tests on required environments (local, cloud, virtual machines)
- Execute in **parallel** to save time
- Schedule tests for continuous execution

8. Test Reporting & Analysis

- Generate logs and reports (HTML, XML, dashboards)
- Track failures with screenshots, logs, or video recordings
- Integrate with defect tracking tools (JIRA, Bugzilla)

9. Maintenance

- Update scripts when application changes
- Remove obsolete tests

- Refactor to improve efficiency

🔹 9.3 Best Practices in Test Automation

- Start small, automate gradually
- Prioritize high-value test cases
- Maintain proper folder structure
- Keep test scripts independent
- Implement CI/CD for continuous testing
- Regularly review & refactor test suites

🔹 9.4 Challenges in Test Automation

- High initial investment
- Frequent script maintenance due to UI changes
- Choosing the wrong tool/framework
- Lack of skilled resources
- Data management issues

🔹 9.5 Interview Questions

Q1. What is the Test Automation Process?
👉 A structured approach that includes feasibility analysis, tool selection, planning, scripting, execution, reporting, and maintenance to automate test cases effectively.

Q2. Which test cases should not be automated?
👉 One-time test cases, exploratory testing, and rapidly changing UI test cases.

Q3. How do you decide which automation tool to use?
👉 Based on project requirements, technology stack, budget, integration support, and team skill set.

Q4. Why is maintenance important in automation?
👉 Because applications evolve, and outdated scripts may cause false failures. Maintenance ensures test reliability.

Chapter 10: Types of Software Testing

1. Introduction

Software testing is not a "one-size-fits-all" process. Different types of testing ensure that the product works as expected under various conditions. Choosing the right type of testing depends on:

- **Project requirements**
- **Development stage**
- **Time & budget**
- **Risk level**

---

2. Broad Categories of Software Testing

Software Testing can broadly be divided into **two categories**:

A. Manual vs Automation Testing

- **Manual Testing** → Performed by human testers without automation tools.
- **Automation Testing** → Uses scripts and tools (like Selenium, JMeter, Appium).

B. Functional vs Non-Functional Testing

- **Functional Testing** → Validates *what* the system does (features & business requirements).
- **Non-Functional Testing** → Validates *how well* the system performs (speed, scalability, security).

---

3. Types of Software Testing (Detailed)

Functional Testing Types

1. **Unit Testing**
    - Tests individual components/modules in isolation.
    - Usually done by developers.
    - Example: Checking if a login function correctly validates user credentials.
2. **Integration Testing**

- Tests how modules interact with each other.
- Example: Login module working correctly with the database.

3. **System Testing**
    - Validates the complete system as per requirements.
    - Example: End-to-end testing of an e-commerce site (search → cart → payment).
4. **Sanity Testing**
    - Quick checks after minor changes.
    - Example: Testing if login still works after updating the password reset module.
5. **Smoke Testing**
    - "Build Verification Testing" – checks if the build is stable.
    - Example: After a new build, checking if app launches and main screens load.
6. **Regression Testing**
    - Ensures old features still work after new changes.
    - Example: After adding "wishlist" feature, checking cart and payment still work.
7. **User Acceptance Testing (UAT)**
    - Done by clients/end users before release.
    - Ensures product meets business needs.

Non-Functional Testing Types

1. **Performance Testing**
    - Measures response time, throughput, resource usage.
    - Example: Website should load within 2 seconds for 1000 users.
2. **Load Testing**
    - Tests system under expected user load.

    - Example: Checking if an e-learning site can handle 5000 students logging in.

3. **Stress Testing**
    - Pushes system beyond normal limits to check breaking point.
    - Example: Flooding a server with requests until it crashes.
4. **Security Testing**
    - Identifies vulnerabilities and ensures data protection.
    - Example: Testing for SQL Injection, XSS attacks.
5. **Compatibility Testing**
    - Ensures software works across browsers, OS, and devices.
    - Example: Website should work on Chrome, Firefox, Safari, Android, iOS.
6. **Recovery Testing**
    - Tests system's ability to recover after crash/failure.
    - Example: Bank system should restore after power failure.

---

4. Comparison Table

Functional Testing	**Non-Functional Testing**
Focus: What the system does	Focus: How the system works
Example: Login works or not	Example: Login page loads in 2s
Based on requirements	Based on performance/quality
UAT, Regression, Unit	Load, Stress, Security, Recovery

---

5. Real-World Example

Think of a **banking app**:

- **Functional Testing**: Checking if money transfers, deposits, and withdrawals work correctly.
- **Non-Functional Testing**: Checking if transfers happen in < 2 seconds, app is secure against hackers, and it works on both iPhone & Android.

---

6. Common Interview Questions

1. Difference between functional and non-functional testing?
2. When do you perform regression testing?
3. What is the difference between smoke and sanity testing?
4. Give real-time examples of performance vs load vs stress testing.
5. Why is UAT important?

Chapter 11: Unit Testing

11.1 Introduction

Unit Testing is the **first level of software testing** where individual components or modules of a software system are tested in isolation.

- A **unit** is the smallest testable part of an application, such as a function, method, or class.
- Goal: To verify that each unit of the software **works as intended**.

11.2 Characteristics of Unit Testing

✓ Tests are focused on **small, isolated pieces** of code.
✓ Performed by **developers** during the coding phase.
✓ Requires use of **stubs/mocks** to simulate dependencies.
✓ Typically automated using **unit testing frameworks**.

11.3 Process of Unit Testing

1. **Identify unit/module** to be tested.
2. **Write test cases** for each function/method.
3. **Use stubs/mocks** for dependencies.
4. **Execute tests** using a framework (e.g., JUnit, PyTest).
5. **Check expected vs actual results**.
6. **Fix defects** if any.
7. **Repeat until all units pass**.

11.4 Example of Unit Testing (Python)

```
# code to test

def add(a, b):

  return a + b
```

```
# unit test
import unittest

class TestMath(unittest.TestCase):
  def test_add(self):
    self.assertEqual(add(2, 3), 5)

if __name__ == "__main__":
  unittest.main()
```

Here:

- Function add() is the **unit**.
- unittest framework is used to test it.

11.5 Advantages of Unit Testing

- ✅ Catches bugs early in development.
- ✅ Simplifies debugging since errors are localized.
- ✅ Improves code quality and design.
- ✅ Provides documentation for the code.
- ✅ Facilitates changes/refactoring safely.

11.6 Disadvantages of Unit Testing

- ❌ Time-consuming for large systems.
- ❌ Cannot detect integration or system-level issues.
- ❌ Requires skilled developers to write good test cases.

11.7 Best Practices for Unit Testing

- Write **small, focused tests**.

- Cover both **positive and negative scenarios**.
- Use **mocking frameworks** to handle dependencies.
- Follow **AAA pattern**: Arrange → Act → Assert.
- Keep tests **automated** and part of CI/CD pipelines.

11.8 Common Unit Testing Frameworks

- **Java** → JUnit, TestNG
- **Python** → unittest, PyTest
- **C#** → NUnit, MSTest
- **JavaScript** → Jest, Mocha

11.9 Unit Testing vs Other Testing

Aspect	**Unit Testing**	**Integration Testing**	**System Testing**
Scope	Single module/function	Multiple modules together	Whole system
Performed by	Developer	Developer/Testers	Testers
Automation level	High	Medium	Medium/Low
Bug detection stage	Early	Mid-level	Later

11.10 Interview Questions

Q1. What is Unit Testing?

👉 Unit Testing is the process of testing the smallest parts (units) of code in isolation to ensure they work correctly.

Q2. Who performs Unit Testing?

👉 Typically developers perform it during the coding phase.

Q3. What are stubs and mocks in Unit Testing?

👉 Stubs and mocks are **dummy objects** used to simulate external dependencies.

Q4. What are advantages of Unit Testing?

👉 Early bug detection, better design, easier debugging, supports refactoring.

Q5. What is the difference between Unit and Integration Testing?

👉 Unit tests individual components, while integration tests how components work together.

Chapter 13: Regression Testing

13.1 Introduction

- **Definition**:
  Regression Testing is the process of **re-testing a software application after changes (bug fixes, enhancements, or updates) to ensure that existing functionality still works correctly**.
- It ensures **new code doesn't break old code**.

👉 Think of it as a **safety net** that checks whether the software is still stable after modifications.

13.2 Why Regression Testing is Important

1. **Code Changes**: Any new feature or bug fix might affect existing features.
2. **Dependencies**: Changes in one module may impact another linked module.
3. **Unintended Side Effects**: Developers may unknowingly introduce errors.
4. **Continuous Delivery**: In agile/DevOps, frequent updates make regression testing essential.

💡 Example: You fix a login issue → but that fix might break password reset → regression testing catches it.

13.3 When to Perform Regression Testing

- After **bug fixes**.
- After **new functionality is added**.
- After **configuration/environment changes**.
- During **integration testing** and **release cycles**.

13.4 Techniques of Regression Testing

1. **Retest All**: Execute all test cases → very expensive and time-consuming.
2. **Regression Test Selection**: Select a subset of test cases impacted by the changes.

3. **Test Case Prioritization**: Prioritize based on business impact and risk (critical modules first).
4. **Hybrid Approach**: Combination of selection + prioritization.

13.5 Tools for Regression Testing

- **Selenium** – Automating UI regression tests.
- **JUnit/TestNG** – Unit-level regression testing.
- **QTP/UFT** – Functional regression.
- **Jenkins + Automation** – Continuous regression in CI/CD.

13.6 Best Practices in Regression Testing

- Maintain a **regression test suite** (set of reusable test cases).
- Automate frequently repeated regression cases.
- Update test cases whenever features change.
- Perform regression testing **early and continuously** in Agile.

13.7 Example

Suppose an **e-commerce app** adds a "Wishlist" feature.

- **New Test**: Wishlist works properly.
- **Regression Test**: Ensure cart, checkout, and login still work correctly.

13.8 Interview Questions (with Answers)

Q1. What is regression testing?

- Regression testing ensures that new code changes do not negatively impact existing functionality.

Q2. How is regression testing different from retesting?

- **Retesting**: Validates if a specific defect is fixed.
- **Regression Testing**: Checks whether the fix has affected other functionalities.

Q3. Which test cases are selected for regression testing?

- Critical business flows, frequently used features, and high-risk modules.

Q4. Can regression testing be automated?

- Yes, regression testing is highly suitable for automation because it involves repeated execution of the same set of test cases.

Q5. Give an example of regression testing in real life.

- A banking app fixes a bug in the fund transfer module. Regression testing ensures login, balance inquiry, and transaction history are still working.

Chapter 14: Smoke Testing

14.1 Introduction

Smoke Testing, also known as **Build Verification Testing (BVT)**, is a type of software testing performed on the initial builds of a software product to ensure that the **critical functionalities** work correctly.
It is like a "quick health check" of the application before deeper testing is performed.

📌 **Mnemonic:** *"Smoke test = Is it alive?"*

14.2 Definition

- Smoke Testing is a **shallow and wide** testing approach.
- It checks whether the most important functions of the software are working and whether the build is **stable enough for further testing**.
- It is performed after every new build release by developers.

14.3 Purpose of Smoke Testing

1. To **verify build stability**.
2. To **save time** by detecting major issues early.
3. To **avoid wasting effort** on detailed testing if the build is broken.
4. To ensure that **critical functionality** works before regression or functional testing.

14.4 Characteristics of Smoke Testing

- Performed on **new builds**.
- Focuses on **critical paths** (login, database connection, navigation).
- Quick execution (takes minutes to hours).
- Automated or manual.
- Failures → build rejected & sent back to developers.

14.5 Example

Imagine testing an **E-commerce Website** build:

- Can the site be launched in a browser?
- Can a user log in?
- Can a product be searched?
- Can an item be added to the cart?
- Can checkout be initiated?

If any of these **basic flows fail**, the build is rejected.

14.6 Difference: Smoke Testing vs Sanity Testing

Feature	Smoke Testing ✅	Sanity Testing ✅
Purpose	Check build stability	Verify specific bug fixes or features
Level	High-level	Narrow & deep
Performed by	Testers/Developers	Testers
Scope	Covers broad functionality	Focuses on specific module
Automation	Often automated	Mostly manual
When Done	After every build	After bug fix / minor changes

📌 **Mnemonic:**

- **Smoke = Broad check**
- **Sanity = Focused check**

14.7 Advantages of Smoke Testing

- Quickly identifies **major issues**.
- Saves **time & cost**.
- Prevents unstable builds from going to QA.
- Ensures **critical paths** work.

- Easy to automate with CI/CD tools.

14.8 Disadvantages of Smoke Testing

- Does not cover **detailed testing**.
- Cannot guarantee software quality.
- May **miss smaller issues**.
- Limited test coverage.

14.9 Real-World Tools for Smoke Testing

- **Selenium** – Automate smoke test cases.
- **Jenkins** – Run smoke tests on each build.
- **JUnit/TestNG** – Unit-level smoke tests.
- **Postman** – API smoke testing.

14.10 Interview Questions

Q1. What is Smoke Testing?
👉 Smoke Testing is a preliminary test to check if the critical functionalities of an application are working and if the build is stable enough for further testing.

Q2. Why is it called "Smoke Testing"?
👉 Like checking if new hardware "smokes" when first powered on, in software we check if the build is stable without crashing.

Q3. When is Smoke Testing performed?
👉 After receiving a new build from developers, before regression or functional testing.

Q4. Difference between Smoke and Sanity Testing?
👉 Smoke = Build Verification (broad & shallow).
👉 Sanity = Bug Fix Verification (narrow & deep).

Q5. Can Smoke Testing be automated?
👉 Yes, automated smoke tests are common in CI/CD pipelines to quickly validate builds.

Chapter 15: Smoke Testing

15.1 Introduction

Smoke Testing, also called **Build Verification Testing (BVT)**, is a **shallow and wide approach** to testing. It ensures that the **most important and critical functionalities of the application are working** after a new build is released, before moving forward with detailed testing.

It acts like a **pre-check** or **gatekeeper** to decide whether the build is stable enough for further testing.

15.2 Definition

Smoke Testing is a type of software testing performed on a new build to ensure that the basic and critical functionalities of the application are working correctly.

If the smoke test fails, the build is **rejected** and sent back to developers for fixing.

15.3 Key Features of Smoke Testing

- Performed on every **new build or release**.
- Focuses on **critical functionalities only**.
- Ensures **build stability** before detailed testing.
- Can be performed **manually** or **automated**.
- Saves **time and effort** by avoiding testing on unstable builds.

15.4 Objectives of Smoke Testing

- To check if the **major functions of the software are working**.
- To verify whether the build is **stable enough** for further testing.
- To reduce the risk of wasting resources on a **defective build**.
- To detect **showstopper defects** at the earliest stage.

15.5 Process of Smoke Testing

1. **Receive Build** → Developers release a new build.
2. **Deploy Build** → QA team installs and configures the build.
3. **Execute Smoke Tests** → Run predefined critical test cases.
4. **Evaluate Results**:
    - ✅ Pass → Proceed with detailed testing.
    - ❌ Fail → Reject build and send it back to development.

15.6 Example of Smoke Testing

E-commerce Application

- Verify if the application launches successfully.
- Verify if login functionality works.
- Verify if products can be searched.
- Verify if "Add to Cart" and "Checkout" options work.

If any of these fail → Build is rejected.

15.7 Advantages

- Quickly identifies **unstable builds**.
- Saves **time and effort** by preventing wasted testing.
- Increases **confidence** in build stability.
- Can be automated for **CI/CD pipelines**.

15.8 Disadvantages

- Covers only **basic functionalities**, not in-depth.
- Cannot detect **all defects**.
- Sometimes mistaken as a **replacement for detailed testing** (which it is not).

15.9 Difference Between Smoke and Sanity Testing

Aspect	Smoke Testing	Sanity Testing
Purpose	To check build stability.	To check correctness of bug fixes/new features.
Coverage	Broad and shallow (critical features).	Narrow and deep (specific modules).
Performed On	New builds.	Stable builds after changes/fixes.
Objective	Verify stability for further testing.	Verify correctness of specific functionality.

15.10 Interview Questions on Smoke Testing

1. **What is Smoke Testing?**
   → Smoke Testing is a type of testing performed on new builds to verify critical functionalities and decide build stability.
2. **Why is Smoke Testing important?**
   → It helps avoid wasting time on unstable builds by catching critical defects early.
3. **What is another name for Smoke Testing?**
   → Build Verification Testing (BVT).
4. **How is Smoke Testing different from Sanity Testing?**
   → Smoke = Broad check of stability.
   Sanity = Narrow check of correctness after fixes.
5. **Can Smoke Testing be automated?**
   → Yes, especially in **CI/CD pipelines** using tools like Jenkins, Selenium, or JUnit.

15.11 Summary

- Smoke Testing = Build Verification Testing.
- Ensures **basic functionality works**.
- If smoke test fails → Build is **rejected**.
- Helps save **time, effort, and cost** in testing.
- Commonly automated in **Agile & DevOps environments**.

Chapter 16 – Performance Testing Overview

16.1 Introduction

Performance Testing is a type of **non-functional testing** that evaluates the **speed, scalability, stability, and responsiveness** of a software application under a particular workload.

Its goal is not to find defects, but to **ensure that the system meets performance benchmarks** and provides a good **user experience** under expected conditions.

16.2 Objectives of Performance Testing

- ✅ Ensure application **responds quickly**.
- ✅ Identify and **eliminate performance bottlenecks**.
- ✅ Verify **scalability** (can system handle growth in users/data?).
- ✅ Guarantee **stability** under peak load conditions.
- ✅ Validate that application meets **SLA (Service Level Agreement)**.

16.3 Key Performance Parameters

1. **Response Time** → Time taken to respond to a request.
2. **Throughput** → Number of transactions per second (TPS).
3. **Latency** → Delay between request and response.
4. **Scalability** → System's ability to handle increased load.
5. **Resource Utilization** → CPU, memory, network usage.

16.4 Types of Performance Testing

1. **Load Testing** → Check performance under expected user load.
2. **Stress Testing** → Push system beyond its limits.
3. **Spike Testing** → Sudden increase in users/requests.
4. **Endurance Testing (Soak)** → Long-term stability test.

5. **Scalability Testing** → Check how well system scales when hardware/users increase.

16.5 Common Performance Testing Tools

- **Apache JMeter** – Open-source, widely used.
- **LoadRunner** – Commercial tool for large enterprise testing.
- **Gatling** – Developer-friendly tool for load testing.
- **NeoLoad** – For complex performance and DevOps pipelines.

16.6 Performance Testing Process

1. **Requirement Gathering** – Define performance benchmarks.
2. **Test Planning** – Choose scenarios, workloads, tools.
3. **Test Environment Setup** – Create production-like test environment.
4. **Test Script Development** – Record/prepare test scripts.
5. **Execution** – Run performance tests.
6. **Monitoring** – Track CPU, memory, response time, DB performance.
7. **Analysis & Reporting** – Identify bottlenecks, optimize.

16.7 Real-World Example

Imagine an **e-commerce website**:

- During a **Black Friday sale**, the system must support **1 million users** without crashing.
- Performance Testing checks how the site behaves when users browse, search, and purchase simultaneously.

16.8 Advantages

- Ensures **faster and smoother user experience**.
- Detects **system bottlenecks** early.
- Prevents **financial and reputational losses** from downtime.

- Helps with **capacity planning**.

16.9 Disadvantages

- Requires **specialized tools & environment**.
- Can be **time-consuming and costly**.
- Complex for large enterprise systems.

16.10 Interview Questions

Q1. What is performance testing and why is it needed?

✓ To ensure system speed, stability, and scalability under workload.

Q2. What is the difference between load and stress testing?

✓ Load Testing → Normal expected load.

✓ Stress Testing → Beyond system's capacity.

Q3. Which tools are commonly used for performance testing?

✓ JMeter, LoadRunner, Gatling, NeoLoad.

Q4. What are key performance parameters?

✓ Response time, throughput, scalability, latency, resource utilization.

Q5. What is the difference between functional and performance testing?

✓ Functional testing checks **correctness of features**, performance testing checks **speed, stability, scalability**.

Chapter 17: Load Testing

1. Introduction

Load Testing is a type of **Performance Testing** where the application is tested under **expected user load** to check how it behaves in normal and peak conditions. The main goal is to **identify bottlenecks** before the system goes live.

---

2. Definition

Load Testing is the process of applying expected workload (number of users, requests, or data volume) to a software system to evaluate its performance, response time, throughput, and resource utilization.

---

3. Key Objectives of Load Testing

1. **Measure performance** under normal and peak usage.
2. **Ensure stability** of the system with expected traffic.
3. **Identify bottlenecks** in CPU, memory, or database.
4. **Determine response time** for different transactions.
5. **Verify scalability** (how system handles increased users).

---

4. Example

Imagine an **E-commerce website** like Amazon:

- Normal expected users = **10,000 concurrent shoppers**.
- During a sale event, the system must handle this load without **slow checkout, cart issues, or crashes**.
- Load testing simulates these users to ensure the system remains stable.

---

5. Characteristics of Load Testing

- Always done in a **production-like environment**.
- Focuses on **volume of users** and **transactions per second (TPS)**.

- Metrics observed:
    - Response time
    - Throughput
    - Error rate
    - Server utilization

---

6. Tools for Load Testing

- **Apache JMeter**
- **LoadRunner (Micro Focus)**
- **Gatling**
- **BlazeMeter**
- **Locust (Python-based)**

---

7. Load Testing Process

1. **Identify business scenarios** (e.g., login, checkout, payment).
2. **Define load conditions** (e.g., 5k, 10k, 20k users).
3. **Prepare test scripts** (using JMeter, LoadRunner, etc.).
4. **Execute load tests** with monitoring tools.
5. **Analyze results** (response time, bottlenecks, resource usage).
6. **Report findings** and suggest performance improvements.

---

8. Benefits of Load Testing

- Detects **performance bottlenecks** early.
- Improves **user satisfaction** (no slowdowns).
- Helps in **capacity planning**.
- Ensures **reliability and scalability**.

---

9. Interview Questions

Q1. What is Load Testing?
Load Testing checks how an application performs under expected user load to ensure stability and response time.

Q2. How is Load Testing different from Stress Testing?

- **Load Testing** = Expected load.
- **Stress Testing** = Beyond expected load (system breaking point).

Q3. Name some load testing tools.
JMeter, LoadRunner, Locust, Gatling, BlazeMeter.

Q4. What key metrics are monitored during Load Testing?
Response time, throughput, CPU/Memory usage, TPS, error rate.

Chapter 18 – Load Testing

1. Introduction

Load Testing is a **type of performance testing** that determines how a software application behaves when subjected to a specific expected load.
It helps verify:

- System's **response time**
- **Throughput** (transactions per second)
- **Resource utilization** (CPU, memory, database, network)
- Whether the system meets **performance benchmarks** under normal and peak conditions.

📌 In simple terms: *Load testing checks how well your software works when many users use it at the same time.*

---

2. Objectives of Load Testing

- Ensure the system can handle **expected user load**.
- Identify **bottlenecks** in hardware, software, or network.
- Verify system stability and **scalability**.
- Provide metrics for **capacity planning**.
- Avoid performance issues after deployment.

---

3. Features of Load Testing

- Conducted under **controlled conditions**.
- Focuses on **simulating concurrent users/transactions**.
- Measures **system performance under real-world usage**.
- Helps in **early defect detection** related to performance.

---

4. Tools for Load Testing

Some popular load testing tools:

- **Apache JMeter** – Open-source, widely used.
- **LoadRunner (Micro Focus)** – Enterprise-grade tool.
- **NeoLoad** – Commercial load testing tool.
- **Locust** – Python-based open-source tool.
- **Gatling** – Developer-friendly load testing.

---

5. Load Testing Process

1. **Identify performance criteria**
   (e.g., response time should be < 2 seconds for 1000 users).
2. **Plan load scenarios**
   (number of users, type of transactions).
3. **Prepare test environment**
   (similar to production setup).
4. **Execute load tests**
   (using automation tools).
5. **Monitor system resources**
   (CPU, memory, DB connections).
6. **Analyze results**
   (check if KPIs are met).
7. **Report and optimize**
   (recommend fixes for bottlenecks).

---

6. Example

- Suppose an **E-commerce website** expects **10,000 concurrent users** during a festive sale.
- A load test simulates this number of users browsing, adding items to cart, and checking out.
- If the **response time increases beyond 5 seconds**, optimization is needed in **database queries or caching**.

---

7. Advantages

- Identifies performance bottlenecks before production.
- Improves **user satisfaction** by ensuring smooth experience.
- Helps in **capacity planning**.
- Prevents **system crashes** in real scenarios.
- Enhances **scalability and reliability**.

---

8. Challenges

- Requires **realistic test environment** (costly).
- Simulating **real user behavior** can be complex.
- Analyzing huge performance data can be challenging.

---

9. Interview Questions

Q1. What is load testing?
👉 Load testing is a type of performance testing used to check how a system behaves under expected user load conditions.

Q2. How is load testing different from stress testing?
👉 Load testing checks performance under expected load, while stress testing checks behavior under extreme/unexpected load.

Q3. Name some load testing tools.
👉 Apache JMeter, LoadRunner, NeoLoad, Locust, Gatling.

Q4. What parameters are measured during load testing?
👉 Response time, throughput, CPU/memory usage, error rate, DB performance.

Q5. Why is load testing important?
👉 To ensure system stability, detect bottlenecks, and provide a smooth user experience under real-world conditions.

---

10. Key Points to Remember

- Load testing = **Expected load testing**.
- Goal: Measure **performance, stability, and scalability**.
- Tools: **JMeter, LoadRunner, Locust**.

- Critical for **websites, mobile apps, APIs, enterprise software**.

Chapter 19: Mobile Testing

1. Introduction

Mobile Testing is the process of verifying and validating mobile applications (apps) for functionality, usability, performance, and security on different devices, operating systems, and network conditions.
With the explosive growth of smartphones and tablets, ensuring that mobile apps deliver a seamless experience has become critical.

---

2. Why Mobile Testing is Important?

- Different **Operating Systems (OS):** Android, iOS, Windows, etc.
- Device **fragmentation:** Multiple devices with different screen sizes, resolutions, and hardware.
- **Network variability:** 2G, 3G, 4G, 5G, and Wi-Fi.
- **App performance:** Must be smooth and responsive even under stress.
- **User experience:** Mobile apps should be user-friendly, intuitive, and secure.

👉 Poorly tested apps lead to bad user experience → uninstalls → negative reviews.

---

3. Types of Mobile Applications

1. **Native Apps** – Developed for a specific platform (Android/iOS). Example: WhatsApp.
2. **Web Apps** – Accessed via a browser, not installed. Example: m.flipkart.com.
3. **Hybrid Apps** – Combination of native and web apps. Example: Instagram, Uber.

---

4. Mobile Testing Types

1. **Functional Testing** – Ensures the app functions as expected.
2. **Usability Testing** – Ensures app is user-friendly.
3. **Compatibility Testing** – Tests on multiple devices, OS versions, browsers.
4. **Performance Testing** – Load, stress, stability under different conditions.
5. **Security Testing** – Protects user data and prevents hacking.

6. **Interrupt Testing** – Handles interruptions (calls, messages, notifications).
7. **Installation/Uninstallation Testing** – Smooth app installation, upgrade, and removal.
8. **Network Testing** – Tests app under various network conditions (offline/poor connectivity).

---

5. Mobile Testing Approaches

- **Manual Testing**
    - Testers execute test cases manually.
    - Best for usability and exploratory testing.
- **Automation Testing**
    - Uses tools like Appium, Espresso, Robotium, Selendroid.
    - Reduces time and increases coverage.

---

6. Mobile Testing Tools

- **Appium** – Open-source, supports Android & iOS.
- **Espresso** – Google's tool for Android UI testing.
- **XCUITest** – Apple's UI testing tool for iOS.
- **Robotium** – Android test automation framework.
- **TestComplete** – Supports automated testing of mobile, web, and desktop apps.

---

7. Mobile Testing Challenges

- Huge device fragmentation.
- OS upgrades (frequent Android & iOS updates).
- Battery consumption & memory usage.
- Network bandwidth differences.
- Maintaining test scripts across platforms.

---

8. Best Practices in Mobile Testing

- Test on **real devices** instead of only emulators.
- Prioritize testing on **popular devices and OS versions**.
- Use cloud-based device farms (BrowserStack, Sauce Labs).
- Automate repetitive test cases.
- Always test **security** and **data privacy**.

---

9. Interview Questions

Q1. What are the types of mobile apps?

👉 Native, Web, Hybrid.

Q2. What is the difference between Emulator and Simulator?

- **Emulator** – Mimics hardware + software (Android).
- **Simulator** – Mimics software only (iOS).

Q3. Which tools are used for mobile testing?

👉 Appium, Espresso, XCUITest, Robotium.

Q4. What are the key challenges in mobile testing?

👉 Device fragmentation, OS updates, network variability, battery issues.

Q5. Why is interrupt testing important?

👉 Ensures app resumes properly after phone calls, SMS, or notifications.

Chapter 20 – Agile Testing

1. Introduction to Agile Testing

- **Definition**:
  Agile Testing is a **software testing practice** that follows the **principles of Agile software development**. Testing is not a separate phase but is **continuous and integrated** throughout the development cycle.
- **Key Idea**:
  Instead of testing after coding (traditional models), Agile testing happens **parallel to development**, ensuring **faster feedback, better collaboration, and high-quality products**.
- **Agile Manifesto Principles in Testing**:
  1. Customer collaboration over contract negotiation
  2. Working software over comprehensive documentation
  3. Responding to change over following a fixed plan
  4. Individuals and interactions over processes and tools

📌 **Example**: In Scrum, testers work closely with developers during each sprint. Test cases are written for **user stories**, and testing is done incrementally.

---

2. Characteristics of Agile Testing

- **Continuous Testing** – Testing begins from the start and continues throughout.
- **Customer Involvement** – Clients are part of requirement discussions and acceptance testing.
- **Teamwork** – Developers, testers, and business analysts collaborate closely.
- **Quick Feedback** – Faster detection of defects, reducing cost and effort.
- **Less Documentation** – Focus is on working software and test automation scripts instead of lengthy test plans.
- **Adaptability** – Testers adapt quickly to changing requirements.

---

3. Agile Testing Life Cycle

Unlike the **Waterfall Testing Life Cycle**, Agile testing follows an **iterative life cycle**:

1. **Planning** – Define user stories, acceptance criteria, and test strategy.
2. **Designing Tests** – Create test cases/scenarios (often BDD or ATDD style).
3. **Coding + Unit Testing** – Developers write code, while testers prepare automation scripts.
4. **Execution** – Continuous integration (CI) ensures builds are tested automatically.
5. **Defect Reporting** – Bugs are logged immediately and fixed within the sprint.
6. **Review & Feedback** – Regular retrospectives ensure continuous improvement.

---

4. Agile Testing Quadrants

Agile testing is divided into **4 quadrants** (by Brian Marick):

- **Quadrant 1: Technology-facing, support the team**
    - Unit tests, Component tests, Automated testing.
- **Quadrant 2: Business-facing, support the team**
    - Functional tests, Examples, Prototypes, Story tests.
- **Quadrant 3: Business-facing, critique the product**
    - Exploratory testing, Usability testing, Acceptance testing.
- **Quadrant 4: Technology-facing, critique the product**
    - Performance testing, Load testing, Security testing.

📌 **Mnemonic to remember Quadrants**:
👉 **"Team Builds, Team Validates, User Checks, System Challenges"**

---

5. Agile Testing Methods

- **Scrum Testing** – Testers work within sprints alongside developers.
- **Behavior Driven Development (BDD)** – Writing test scenarios in natural language (Given-When-Then format).
- **Test Driven Development (TDD)** – Writing tests before writing actual code.
- **Acceptance Test Driven Development (ATDD)** – Writing acceptance tests with input from customers, developers, and testers.
- **Exploratory Testing** – Performing unscripted testing for better coverage.

---

6. Roles & Responsibilities of Agile Testers

- Participate in sprint planning & daily standups.
- Write acceptance criteria and user story test cases.
- Automate regression tests.
- Ensure CI/CD pipelines are validated.
- Collaborate with developers to prevent defects instead of just detecting them.
- Perform exploratory testing.

---

7. Advantages of Agile Testing

✓ Faster feedback loop
✓ Early bug detection reduces cost
✓ High customer satisfaction
✓ Continuous delivery of working software
✓ Improved collaboration

---

8. Challenges in Agile Testing

❌ Less documentation may cause knowledge gaps
❌ Frequent requirement changes
❌ High pressure to deliver quickly
❌ Test automation setup is time-consuming
❌ Coordination with distributed teams

---

9. Agile Testing Tools

- **Project Management** – JIRA, Trello
- **Automation** – Selenium, Cypress, TestNG, JUnit
- **CI/CD** – Jenkins, GitHub Actions, GitLab CI
- **API Testing** – Postman, REST Assured
- **Performance** – JMeter

---

10. Interview Questions on Agile Testing

Q1. What is Agile Testing?
Agile testing is a testing practice that follows agile principles. It is continuous, customer-focused, and integrated with development.

Q2. How is Agile testing different from traditional testing?

- Agile: Continuous, less documentation, collaboration-based.
- Traditional: Sequential, heavy documentation, testing after development.

Q3. What is TDD, BDD, ATDD?

- TDD: Write tests before code.
- BDD: Write tests in natural language (Given-When-Then).
- ATDD: Acceptance tests written with input from customers, developers, testers.

Q4. What are the Agile Testing Quadrants?
Quadrant 1: Unit tests → Quadrant 2: Functional tests → Quadrant 3: Exploratory tests → Quadrant 4: Non-functional tests.

Q5. What are the roles of a tester in Agile?
Participating in planning, writing acceptance criteria, automating regression tests, ensuring CI/CD, and performing exploratory testing.

✅ **Summary (Memory Tip – "FAST")**

- **F** – Feedback is quick
- **A** – Adapt to change
- **S** – Sprint-based testing
- **T** – Team collaboration

Chapter 21: Cross Browser Testing

1. Introduction

Cross Browser Testing (CBT) is a type of software testing used to ensure that a web application or website works correctly across different web browsers, operating systems, and devices.
Since users access applications through multiple browsers (like Chrome, Firefox, Safari, Edge, Opera), CBT helps identify inconsistencies in functionality, layout, and performance.

---

2. Why Cross Browser Testing is Important?

- **Browser Compatibility Issues** – Each browser interprets HTML, CSS, and JavaScript differently.
- **User Experience** – To ensure all users get the same seamless experience.
- **Market Coverage** – Different users prefer different browsers and devices.
- **Business Impact** – A website that doesn't render properly in popular browsers may lose traffic and customers.

---

3. What to Test in Cross Browser Testing?

1. **UI/UX Consistency**
    - Layout, alignment, colors, fonts, and images.
    - Responsiveness on different screen sizes.
2. **Functionality**
    - Forms, buttons, links, dropdowns.
    - Navigation flow, login/logout, search.
3. **Performance**
    - Page load time across browsers.
    - Rendering speed and responsiveness.
4. **Browser-Specific Features**
    - JavaScript events, CSS styles, animations.

- Plug-ins, media playback, and APIs.

---

4. Popular Browsers and Platforms to Test

- **Browsers**: Chrome, Firefox, Safari, Edge, Internet Explorer (legacy), Opera.
- **Devices**: Desktops, tablets, mobile phones.
- **Operating Systems**: Windows, macOS, Linux, Android, iOS.

---

5. Approaches to Cross Browser Testing

1. **Manual Testing**
   - Testers check the application on different browsers manually.
   - Best for small projects.
   - **Limitation**: Time-consuming and error-prone.
2. **Automated Testing**
   - Using automation tools/scripts to run test cases on multiple browsers.
   - Fast, scalable, and reliable.

---

6. Tools for Cross Browser Testing

1. **Selenium WebDriver** – Open-source tool for automation across browsers.
2. **BrowserStack** – Cloud-based platform for live and automated browser testing.
3. **Sauce Labs** – Cloud testing tool with wide browser/device support.
4. **LambdaTest** – Provides real-time cross browser testing.
5. **CrossBrowserTesting.com** – Used for both live and automated testing.

---

7. Best Practices

- Identify **target browsers** based on user analytics.
- Prioritize **popular devices** (e.g., Chrome on Android, Safari on iOS).
- Automate regression testing with **Selenium + BrowserStack/Sauce Labs**.
- Use **responsive design testing** for different screen sizes.

- Maintain **browser compatibility checklists** for QA teams.

---

8. Interview Pointers

- **Q: What is Cross Browser Testing?**
  Testing an application across multiple browsers, devices, and OS to ensure consistency.
- **Q: Why is Cross Browser Testing required?**
  To ensure consistent user experience, avoid browser-specific issues, and increase reach.
- **Q: Name a few tools for Cross Browser Testing.**
  Selenium, BrowserStack, Sauce Labs, LambdaTest.
- **Q: What is the difference between Manual and Automated CBT?**
  Manual involves human effort across browsers, while automated uses tools to execute tests faster.

---

9. Quick Mnemonic for Remembering

"BROWSERS" → Key focus areas in Cross Browser Testing

- **B** – Browser coverage
- **R** – Responsiveness
- **O** – OS compatibility
- **W** – Web elements (buttons, forms)
- **S** – Speed & performance
- **E** – Errors (JS/CSS issues)
- **R** – Rendering issues
- **S** – Scalability with automation

Chapter 22: Cross Browser Testing

22.1 Introduction

Cross Browser Testing (CBT) is a type of **non-functional testing** that ensures a web application works correctly across multiple **browsers, operating systems, and devices**.
Different browsers interpret HTML, CSS, and JavaScript differently, which may cause **inconsistent behavior** in how web pages are displayed or how functionalities work.

Example: A button might look perfect in **Google Chrome** but appear misaligned in **Safari** or **Internet Explorer**.

22.2 Why Cross Browser Testing is Important

1. **Browser Diversity** – Users access websites from Chrome, Firefox, Safari, Edge, Opera, etc.
2. **Device Diversity** – Desktop, mobile, tablets, and smart TVs have different rendering engines.
3. **Market Share Variation** – Chrome leads globally, but Safari dominates on iOS, and Edge has enterprise usage.
4. **Consistency** – Ensures a **uniform user experience** across platforms.
5. **Accessibility** – Prevents users from abandoning due to broken layouts or features.

22.3 When to Perform Cross Browser Testing

- **Before Major Releases** → To validate design and features across browsers.
- **After UI Changes** → CSS or JavaScript updates may break compatibility.
- **Responsive Testing** → For mobile and tablet compatibility.
- **Regression Testing** → After bug fixes or feature changes.

22.4 What to Test in Cross Browser Testing

1. **Functionality**
    - Buttons, forms, dropdowns, navigation menus.

- JavaScript features (popups, AJAX calls).

2. **Design & Layout**
    - Font rendering, alignment, spacing, images.
    - Responsiveness (mobile, tablet, desktop).
3. **Performance**
    - Page load speed on different browsers.
4. **Compatibility**
    - CSS3/HTML5 feature support.
    - Plug-in and extension handling.

22.5 Tools for Cross Browser Testing

1. **BrowserStack** – Cloud-based, supports real devices and browsers.
2. **Sauce Labs** – Automates CBT with Selenium and Appium.
3. **LambdaTest** – Cloud testing with 2000+ browsers.
4. **CrossBrowserTesting.com** – Manual and automated CBT.
5. **Selenium Grid** – Open-source, for parallel browser testing.

22.6 Cross Browser Testing Process

1. **Requirement Analysis**
    - Identify target browsers/devices based on user base.
2. **Test Case Preparation**
    - Define functionality, UI, and compatibility scenarios.
3. **Environment Setup**
    - Use tools (BrowserStack, Selenium Grid).
4. **Test Execution**
    - Run manual/automated tests across browsers.
5. **Defect Reporting**
    - Document browser-specific bugs with screenshots.

6. **Regression Testing**
    - Re-run after fixes to ensure consistency.

22.7 Challenges in Cross Browser Testing

- Huge **number of browsers and versions** to cover.
- Frequent **browser updates**.
- **Device fragmentation** (Android vs iOS).
- Maintaining test environments.
- High cost if using real devices.

22.8 Best Practices

1. Test on **most used browsers** first (based on analytics).
2. Use **cloud-based CBT tools** instead of maintaining labs.
3. Automate repetitive regression tests using Selenium/Appium.
4. Focus on **critical functionalities and layouts**.
5. Keep **test scope realistic** – do not test every possible version.

22.9 Interview Questions

1. What is Cross Browser Testing and why is it important?
2. Name three popular tools for CBT.
3. How is CBT different from compatibility testing?
4. When should you perform CBT?
5. What are the major challenges faced in CBT?
6. Explain how Selenium can be used for cross browser testing.

Chapter 23: Cross Browser Testing

23.1 Introduction

Cross Browser Testing (CBT) is the process of verifying whether a web application or website works as expected across different **browsers, browser versions, operating systems, and devices**.

- A website may look perfect in **Google Chrome** but may break in **Internet Explorer** or **Safari**.
- Different browsers use different rendering engines (e.g., Chrome → Blink, Firefox → Gecko, Safari → WebKit).
- Hence, CBT ensures **consistency, functionality, and performance** across platforms.

🎯 23.2 Why Cross Browser Testing is Important?

1. **Browser Diversity** → Users access sites from Chrome, Firefox, Safari, Edge, Opera, etc.
2. **Device Diversity** → Desktop, tablet, and mobile devices all render differently.
3. **Operating System Dependency** → Windows, macOS, Linux, Android, iOS.
4. **UI/UX Consistency** → Ensures the design, fonts, colors, layouts remain uniform.
5. **Functionality Assurance** → Buttons, forms, navigation menus should behave consistently.

📌 Example:

- A login button may work in Chrome but not respond in Safari due to CSS/JS compatibility issues.

🛠️ 23.3 What to Test in Cross Browser Testing?

- ✅ **UI Testing** → Alignment, fonts, images, responsiveness.
- ✅ **Functional Testing** → Forms, buttons, navigation, dropdowns.
- ✅ **Performance Testing** → Page load speed, responsiveness.
- ✅ **Compatibility Testing** → CSS styles, JavaScript functions, cookies, sessions.

- ✅ **Mobile Browser Testing** → Responsive design & viewport scaling.

🖥️ 23.4 Approaches to Cross Browser Testing

1. Manual Cross Browser Testing

- Install different browsers on your system.
- Open the application and test features manually.
- ✅ Advantage → Useful for quick checks.
- ❌ Limitation → Time-consuming, error-prone, difficult for large apps.

2. Automated Cross Browser Testing

- Use automation tools to test across multiple browsers simultaneously.
- Tools like **Selenium Grid, BrowserStack, LambdaTest**.
- ✅ Advantage → Fast, scalable, accurate.
- ❌ Limitation → Requires setup, scripting knowledge.

⚙️ 23.5 Tools for Cross Browser Testing

🔹 1. Selenium Grid

- Open-source automation tool.
- Supports parallel execution across browsers.
- Requires Selenium scripts.

🔹 2. BrowserStack

- Cloud-based testing platform.
- Provides real browsers & devices for testing.
- No installation required.

🔹 3. LambdaTest

- Cloud-based platform.
- 3000+ browsers, OS combinations.
- Supports screenshots, responsive testing.

🔹 **4. Sauce Labs**

- Cloud testing tool.
- Provides automated testing for web & mobile apps.

📋 **23.6 Challenges in Cross Browser Testing**

1. **Browser & Version Explosion** → Too many versions to test.
2. **Device Fragmentation** → Many devices (Android, iOS, tablets).
3. **Rendering Issues** → CSS & JavaScript differences.
4. **Performance Variations** → Speed differences across browsers.
5. **Testing Cost** → Paid tools may be expensive.

✅ **23.7 Best Practices**

- Test on **most popular browsers** (Chrome, Safari, Edge, Firefox).
- Use **responsive design frameworks** (Bootstrap, Tailwind).
- Automate repetitive tests using Selenium/BrowserStack.
- Maintain a **Browser Compatibility Matrix**.
- Regularly update scripts for new browser versions.

🎤 **23.8 Interview Questions**

1. **What is Cross Browser Testing and why is it important?**
   👉 Ensures a web app works consistently across browsers, devices, and OS.
2. **What are the challenges in Cross Browser Testing?**
   👉 Too many browsers/versions, rendering issues, device fragmentation.
3. **Which tools are used for Cross Browser Testing?**
   👉 Selenium Grid, BrowserStack, LambdaTest, Sauce Labs.
4. **What is the difference between Manual & Automated Cross Browser Testing?**
   👉 Manual → slow, human-driven. Automated → fast, scalable, reliable.
5. **How do you select browsers for testing?**
   👉 Based on target audience, analytics data, and market share of browsers.

Chapter 23: JMeter for Jenkins

23.1 Introduction

Performance testing is critical to ensure that applications can handle expected loads. **Apache JMeter** is widely used for performance and load testing, while **Jenkins** is a popular automation server for Continuous Integration/Continuous Delivery (CI/CD).

Integrating JMeter with Jenkins allows performance tests to run automatically as part of the CI/CD pipeline, ensuring that every new build is tested for scalability and reliability.

23.2 Why Integrate JMeter with Jenkins?

- ✅ **Automation** – Performance tests run automatically after builds.
- ✅ **Consistency** – Same test scripts execute across environments.
- ✅ **Faster Feedback** – Developers get immediate performance reports.
- ✅ **Scalability** – Run tests on multiple agents for large loads.
- ✅ **Continuous Performance Testing** – Ensures applications remain stable under frequent updates.

23.3 Steps to Integrate JMeter with Jenkins

Step 1: Install JMeter

- Download JMeter from Apache JMeter website.
- Unzip and configure the **bin/jmeter.bat (Windows)** or **jmeter.sh (Linux/Mac)**.

Step 2: Install Jenkins

- Download Jenkins from jenkins.io.
- Run java -jar jenkins.war.
- Access Jenkins at: http://localhost:8080.

Step 3: Install Jenkins Plugins

In Jenkins:

- Go to **Manage Jenkins → Manage Plugins**.
- Install:

- **Performance Plugin** (to visualize JMeter results).
- **HTML Publisher Plugin** (for detailed reports).

Step 4: Configure Jenkins Job

1. Create a **New Freestyle Project**.
2. In **Build Steps**, add:
3. jmeter -n -t test_plan.jmx -l results.jtl -e -o HTMLReport
   - -n → Non-GUI mode.
   - -t → JMeter test plan file.
   - -l → Log results.
   - -e & -o → Generate HTML report.

Step 5: Publish JMeter Reports

- Go to **Post-build Actions → Publish Performance Test Result Report**.
- Upload results.jtl file.
- Jenkins will display performance graphs.

23.4 Example JMeter-Jenkins Workflow

1. Developer commits code → Git triggers Jenkins build.
2. Jenkins builds application and deploys it to a test environment.
3. Jenkins runs JMeter scripts (test_plan.jmx) against the application.
4. JMeter generates results (results.jtl) and HTML reports.
5. Jenkins displays performance metrics (response time, errors, throughput).
6. If performance fails (e.g., response > 3 sec), Jenkins marks build as failed.

23.5 Best Practices

- ✔ Always run JMeter in **non-GUI mode** for performance.
- ✔ Store JMeter test scripts (.jmx) in version control (Git).
- ✔ Automate test execution after each code commit.
- ✔ Use **distributed JMeter testing** for heavy load.

- ✔ Set performance thresholds in Jenkins to auto-fail builds if criteria are not met.

23.6 Interview Questions

Q1. Why integrate JMeter with Jenkins?

➡ To automate performance testing in CI/CD and get real-time feedback.

Q2. Which plugins are required for JMeter in Jenkins?

➡ Performance Plugin, HTML Publisher Plugin.

Q3. What is the command to run JMeter in non-GUI mode?

➡ jmeter -n -t test_plan.jmx -l results.jtl -e -o HTMLReport

Q4. How can you fail a Jenkins build based on JMeter results?

➡ Set thresholds in the Performance Plugin (e.g., fail if >10% errors or response >3 sec).

Q5. Can Jenkins run distributed JMeter tests?

➡ Yes, using multiple slave agents.

Interview questions

✅ Set 1: Basic Manual Testing Interview Questions

Q1. What is Software Testing?
Answer:
Software testing is the process of verifying and validating whether a software application meets the requirements and works as expected without defects.
👉 *Impressive Point:* It ensures quality, reliability, and user satisfaction.

Q2. Difference between Verification and Validation?
Answer:

- **Verification** → "Are we building the product right?" (Checking design, documents, code).
- **Validation** → "Are we building the right product?" (Checking actual software against requirements).
  👉 *Impressive Point:* Verification is static, Validation is dynamic.

Q3. What is the difference between QA and QC?
Answer:

- **QA (Quality Assurance):** Process-oriented, prevents defects (focuses on how software is built).
- **QC (Quality Control):** Product-oriented, finds defects (focuses on testing the actual product).
  👉 *Impressive Point:* QA = Prevention, QC = Detection.

Q4. What are Functional and Non-Functional Testing?
Answer:

- **Functional Testing:** Checks *what* the system does (login, search, payment).
- **Non-Functional Testing:** Checks *how well* the system performs (speed, security, scalability).
  👉 *Impressive Point:* Functional ensures features work, Non-functional ensures performance and reliability.

Q5. What is the difference between Smoke Testing and Sanity Testing?
Answer:

- **Smoke Testing:** Quick check to see if the build is stable for deeper testing.
- **Sanity Testing:** Narrow and deep testing to verify specific bug fixes or new features.
  👉 *Impressive Point:* Smoke = Build Verification, Sanity = Change Verification.

✅ **Set 2: Scenario-Based Tricky Manual Testing Interview Questions**

Q6. If the login page accepts both valid and invalid users, how will you test it?
Answer:
I will test with:

- Valid credentials (expected: successful login).
- Invalid username/password (expected: error message).
- Blank fields (expected: validation message).
- SQL injection/XSS (expected: secure handling).
  👉 *Impressive Point:* I test not just positive but also negative and security scenarios.

Q7. What will you do if you find a bug just before release?
Answer:

- First, I will **reproduce the bug** and confirm it.
- Then, I will **assess severity & priority** with the team.
- If it is critical, I will **report immediately to stakeholders** to decide whether to fix or delay release.
  👉 *Impressive Point:* I focus on collaboration and impact analysis, not just reporting.

Q8. How do you prioritize test cases?
Answer:
I prioritize based on:

1. **Business impact** (critical features first).

2. **User frequency** (common scenarios tested early).
3. **Defect-prone areas** (past bug history).
   👉 *Impressive Point:* My approach balances **risk + business value**.

Q9. If a developer says it's not a bug, but you believe it is, what will you do?
Answer:
I will provide **clear steps, screenshots, and logs** to show the issue.
If still unclear, I will compare with **requirement documents** or **user expectations**.
👉 *Impressive Point:* I rely on **evidence and requirements**, not arguments.

Q10. How will you test a feature if requirements are missing?
Answer:

- I will **explore the application** to understand behavior.
- I will discuss with **BA/Developers** to clarify.
- I will use **exploratory testing + user perspective** to design cases.
  👉 *Impressive Point:* I can handle uncertainty using **exploratory + communication skills**.

✅ **Set 3: Common Practical Manual Testing Interview Questions**

Q11. What is the difference between Severity and Priority in defects?
Answer:

- **Severity** → Impact of the defect on the system (technical).
- **Priority** → Order in which defect should be fixed (business need).
  👉 Example: Spelling mistake on home page = Low Severity, High Priority.

Q12. What is the difference between Verification and Validation?
Answer:

- **Verification** = Are we building the product right? (reviews, walkthroughs, static testing).
- **Validation** = Are we building the right product? (executing tests, dynamic testing).
  👉 Easy way: Verification = Paper check, Validation = Real execution.

Q13. What is the difference between Functional and Non-Functional Testing?
Answer:

- **Functional Testing** → Tests *what* the system does (login, search, checkout).
- **Non-Functional Testing** → Tests *how* the system performs (performance, security, usability).

Q14. Explain Smoke Testing vs Sanity Testing.
Answer:

- **Smoke Testing** → Basic check to ensure build is stable ("Is it smoking?").
- **Sanity Testing** → Narrow regression testing to check if specific fixes work.
  👉 Mnemonic: Smoke = Build check, Sanity = Fix check.

Q15. What is Regression Testing?
Answer:
Regression Testing ensures that **new code changes haven't broken existing features**.
👉 Example: After fixing "payment bug," test cart, checkout, and invoice again.

✅ **Set 4: Advanced & Real-Time Scenario Questions**

Q16. What is Exploratory Testing?
Answer:
Exploratory Testing means **simultaneous learning, test design, and execution**.
👉 Tester explores the application without predefined test cases, relying on experience and intuition.

Q17. What is Ad-hoc Testing? How is it different from Exploratory Testing?
Answer:

- **Ad-hoc Testing** → Random, unstructured testing without documentation.
- **Exploratory Testing** → Still unstructured but with a goal and learning process.
  👉 Difference: Ad-hoc = Random, Exploratory = Intelligent.

Q18. How do you test a Login Page?
Answer:

- Valid username & password → success.
- Invalid username/password → error message.
- Empty fields → validation error.
- SQL Injection, special characters.
- Password masking.
  👉 Covers both functional + security checks.

Q19. What is the difference between Positive and Negative Testing?
Answer:

- **Positive Testing** → Checking the system with valid inputs (system should work).
- **Negative Testing** → Checking with invalid inputs (system should fail gracefully).

Q20. What would you do if you find a critical defect just before release?
Answer:

- Immediately report it to the team (with severity explained).
- Highlight business impact.
- Discuss with stakeholders whether to fix immediately or delay release.
  👉 Communication + risk assessment is key here.

✅ **Set 5: Tricky & Situational Manual Testing Questions**

Q21. How will you test a feature if the requirements are unclear or missing?
Answer:

- Explore the application to understand its behavior.
- Discuss with BA/Developer to clarify.
- Use **exploratory testing** to identify defects.
  👉 Shows your ability to handle uncertainty and think critically.

Q22. How do you write a good test case?
Answer:
A good test case should have:

- Unique ID and Title
- Clear Steps to execute
- Expected Result
- Actual Result (after execution)
- Pre-conditions & Post-conditions
- Priority & Severity
  👉 Impressive point: Test cases should be **simple, clear, and reusable**.

Q23. How do you handle repeated defects?
Answer:

- Track using defect management tools (Jira, Bugzilla).
- Analyze root cause and discuss with developers.
- Suggest improvements in code or process.
  👉 Shows problem-solving and preventive approach.

Q24. What is the difference between Error, Bug, and Defect?
Answer:

- **Error** → Human mistake in code/design.
- **Bug** → Flaw in software caused by an error.
- **Defect** → Deviation from requirements in final product.
  👉 Easy trick: Error → Developer, Bug → Code, Defect → Product.

Q25. How do you test an application without documentation?
Answer:

- Use **Exploratory Testing** to learn the app behavior.
- Prioritize high-risk modules first.

- Communicate with stakeholders to validate expected behavior.
  👉 Shows independence, analytical skills, and communication.