✅ What is Software Testing?

Software Testing is the process of evaluating and verifying that a software application or system functions as expected. It involves executing software/system components using manual or automated tools to identify bugs, errors, or missing requirements.

In simple terms, **software testing ensures that the software product is reliable, performs well, and meets user requirements.**

✅ Why is Software Testing Necessary?

Software testing is crucial for several reasons:

1. □ **Detecting Bugs Early:**
   - Testing helps identify bugs and issues early in the development cycle, reducing the cost and time of fixing them later.
2. ✅ **Ensures Quality:**
   - Testing verifies that the software works correctly, providing confidence in the product's stability and performance.
3. 🔐 **Security Assurance:**
   - Security testing ensures the software is free from vulnerabilities, especially critical in banking, healthcare, and e-commerce apps.
4. 👤 **Customer Satisfaction:**
   - Delivering a bug-free, smooth user experience increases customer trust and satisfaction.
5. 📋 **Validates Requirements:**
   - Testing confirms that the application meets both functional (what it should do) and non-functional (how it should do it) requirements.
6. 📉 **Reduces Maintenance Costs:**
   - Well-tested software is less likely to fail, reducing future maintenance and support costs.
7. 📈 **Improves Performance:**
   - Testing can identify performance issues (like speed, responsiveness) that can be optimized before release.

✅ Difference Between Verification and Validation:

Aspect	Verification □	Validation □
Definition	Are we building the product **right**?	Are we building the **right** product?
Purpose	To check whether the software meets the specified requirements.	To check whether the software meets the **user's needs** and expectations.
Focus	Process-oriented	Product-oriented
Performed During	Early stages (requirement gathering, design)	After development (post-coding)
Type of Testing	Static Testing (no code execution)	Dynamic Testing (code execution involved)

✅ In Summary:

- **Verification** = *"Did we follow the process correctly?"*
- **Validation** = *"Did we build what the customer actually wanted?"*

✅ What is SDLC? *(Software Development Life Cycle)*

SDLC is the **process followed to develop software**, from idea to deployment and maintenance. It defines **phases** that ensure high-quality, cost-effective software is delivered efficiently.

Phases of SDLC:

1. **Requirement Gathering & Analysis** – Understand what the client needs.
2. **Planning** – Estimate time, cost, and resources.
3. **Design** – Create architecture and UI/UX designs.
4. **Development** – Actual coding of the application.
5. **Testing** – Test the software to find and fix bugs.
6. **Deployment** – Release the software to users.
7. **Maintenance** – Update and fix issues after release.

Goal of SDLC: Deliver functional and reliable software that meets client requirements.

✅ What is STLC? *(Software Testing Life Cycle)*

STLC is a **subset of SDLC** that focuses only on **testing activities**. It defines a sequence of activities to ensure the software is tested systematically and thoroughly.

Phases of STLC:

1. **Requirement Analysis** – Understand what to test from business requirements.
2. **Test Planning** – Define the test strategy, tools, schedule, and resources.
3. **Test Case Design** – Write detailed test cases and test data.
4. **Test Environment Setup** – Prepare hardware/software where testing will happen.
5. **Test Execution** – Run the tests and report bugs.
6. **Test Closure** – Finalize documents, reports, and lessons learned.

Goal of STLC: Ensure the software meets quality standards before release.

❓ SDLC vs STLC – Key Differences

Feature	SDLC	STLC
Full Form	Software Development Life Cycle	Software Testing Life Cycle
Focus	Complete software development	Only software testing process
Involves	Developers, designers, testers	QA testers and test leads
Starts With	Requirement gathering	Requirement analysis for testing

✅ Difference Between Severity and Priority

Aspect	Severity ⚠️	Priority ⏱️
Definition	Describes **how serious** the bug is in terms of functionality or system impact.	Describes **how quickly** the bug should be fixed.
Focus	Technical impact of the defect.	Business urgency of the defect.
Decided By	Usually **Testers** or QA team.	Usually **Product Manager** or **Client** (sometimes with QA input).
Example	App crashes on login → High severity.	If it's a release-critical bug → High priority.

□ Severity Types:

- **Critical** – System crash, data loss, no workaround.
- **Major** – Big functionality broken, workaround exists.
- **Minor** – Small issue, doesn't affect core functions.
- **Trivial** – UI issue, typo, cosmetic glitch.

⏱️ Priority Levels:

- **High Priority** – Fix immediately.
- **Medium Priority** – Fix in the next build or sprint.
- **Low Priority** – Fix whenever possible.

✅ Regression Testing vs Retesting

Feature	Regression Testing 🔁	Retesting 🔄
Definition	Testing to make sure **existing functionality still works** after code changes.	Testing to confirm a **specific defect/bug is fixed.**
Purpose	To detect any **unexpected side effects** of changes.	To verify the **original issue is resolved.**
Focus Area	**Other parts** of the application that might be affected.	**Only the defect area** that was previously failing.
Test Cases Used	Old test cases from the regression suite.	Specific failed test cases from earlier runs.
Execution Priority	Usually run **after retesting.**	Done **immediately after a fix** is deployed.
Automation	Commonly automated (Selenium, TestNG, etc.)	Often done manually (but can be automated too).

□ In Simple Words:

- **Retesting** = *"Bug fixed? Let me confirm."*
- **Regression Testing** = *"Bug fixed? Cool — but did it break anything else?"*

✅ What is a Test Case?

A **Test Case** is a **documented set of actions**, conditions, and inputs used to **verify a specific feature or functionality** of a software application.

🔍 It tells the tester **what to test**, **how to test it**, and **what result to expect**.

✅ Components of a Test Case:

Component	Description
Test Case ID	Unique identifier for the test case (e.g., TC_001)
Test Case Title / Name	A short description of the functionality being tested (e.g., "Login Functionality")
Preconditions	Any setup or state required before executing the test
Test Steps	Step-by-step actions to perform the test
Test Data	Data required to perform the test (e.g., username, password)
Expected Result	The outcome you expect after performing the steps
Actual Result	The actual outcome after executing the test (filled during execution)
Status (Pass/Fail)	Final result of the test case (Pass or Fail)
Priority	Business urgency (High, Medium, Low)
Severity	Technical impact of the issue (if any defect is found)
Remarks/Comments	Additional notes or observations
Tested By / Date	Who performed the test and when

✅ What is a Test Scenario?

A **Test Scenario** is a **high-level description of what to test**. It represents a **real-world use case** or **business flow** that needs to be tested.

🔍 It answers: **"What should be tested?"**

☑ What is a Test Case?

A **Test Case** is a **detailed set of steps**, inputs, and expected results used to verify a specific part of a **test scenario**.

🔍 It answers: **"How should it be tested?"**

🧪 Example to Understand:

🧪 Test Scenario:

👉 *Verify login functionality of the application.*

🧪 Related Test Cases:

Test Case	Description
TC_001	Login with valid email and password
TC_002	Login with invalid password
TC_003	Login with blank fields
TC_004	Login with SQL injection
TC_005	Check "Remember Me" functionality

🧪 Sample Test Cases for Login Page

Here's a variety of **positive** and **negative** test cases covering functional, UI, and validation aspects:

Test Case ID	Test Case Description	Test Steps	Test Data	Expected Result
TC_Login_01	Login with valid credentials	1. Enter valid email & password 2. Click Login	Email: user@example.com Password: 123456	Redirect to Dashboard
TC_Login_02	Login with invalid password	Enter valid email, wrong password	user@example.com / wrongpass	Show error: "Invalid credentials"
TC_Login_03	Login with invalid email format	Enter email without `@`, e.g., "user.com"	Email: user.com / any	Show validation error
TC_Login_04	Login with blank email and password	Leave both fields empty	(blank)	Show validation message: "Email and password required"
TC_Login_05	Login with SQL injection	Enter: `admin' OR '1'='1`	SQL code as input	Show error / prevent login

Test Case ID	Test Case Description	Test Steps	Test Data	Expected Result
TC_Login_06	Check "Remember Me" functionality	1. Check box 2. Login 3. Close & reopen browser	Valid login	Email stays saved
TC_Login_07	Forgot Password link redirects correctly	Click "Forgot Password" link	-	Redirect to Reset page
TC_Login_08	Password field is masked	Type in password field	any	Characters should be masked (••••)
TC_Login_09	Login button disabled if fields are blank	Leave email and password blank	-	Login button remains disabled
TC_Login_10	Error disappears after correct login	1. Try invalid login 2. Try correct credentials	-	Error message clears & user logs in

☑ What is a Bug/Defect?

A **bug** (or **defect**) is any **flaw, error, or unexpected behavior** in a software application that causes it to **produce incorrect or unintended results**.

📌 In simple terms: A **bug** is when the actual result doesn't match the expected result.

🔍 Examples of Bugs:

- Clicking "Login" with valid credentials doesn't redirect to the dashboard.
- Error message shows the wrong information.
- UI overlaps on smaller screen sizes.
- App crashes when you upload a large file.

🐞 How to Report a Bug?

Bugs are typically reported in **bug tracking tools** like **JIRA**, **Bugzilla**, **Redmine**, or even **Excel sheets** in small teams.

☑ 1. Equivalence Partitioning (EP)

Definition:
Equivalence Partitioning divides the input data of a system into **equal and valid partitions (classes)** where test cases are designed for **one value from each partition**, assuming all values in that class behave similarly.

🔍 *"Test one from each group — if one works, others will too."*

✅ 2. Boundary Value Analysis (BVA)

Definition:
Boundary Value Analysis focuses on **testing the values at the boundaries** of input ranges because bugs often occur there.

🔍 *"Most bugs live at the edges."*

🔁 EP vs BVA – Key Differences

Feature	Equivalence Partitioning	Boundary Value Analysis
Focus	Divides input into valid/invalid sets	Focuses on boundary limits of those sets
Test Case Efficiency	Reduces test cases by class	Finds bugs at input edges
Example Range	Age 18–60 → test 30, 15, 65	Age 18–60 → test 17, 18, 19, 59, 60, 61
Type	Generalization	Edge-specific

✅ What is Smoke Testing?

Smoke Testing is a **preliminary check** done to verify that the **basic functionalities of the application are working** and the build is stable enough for deeper testing.

🔍 Also called: **"Build Verification Testing"**

🔧 Example:

- Can you **open the app**?
- Can you **log in**?
- Is the **homepage loading**?

🔥 If it "smokes" (fails), you don't test further!

✅ What is Sanity Testing?

Sanity Testing is a **narrow, deep check** done after a **minor code change or bug fix**, to ensure that **specific functionality is working as expected** and no new bugs are introduced.

🔍 Think of it as **focused re-checking**.

🔧 Example:

- A bug was fixed in the "Add to Cart" button →
  You do **Sanity Testing** to check:
  - Is the "Add to Cart" working?
  - Did it break the cart page?

✅ What is a Defect Life Cycle?

The **Defect Life Cycle** is the **journey a bug goes through** from the moment it is found until it is fixed, retested, and finally closed.

🔍 It defines **how a bug is reported, assigned, fixed, tested, and tracked** through various states.

🔁 Defect Life Cycle Stages

Here's a typical flow of defect states:

1. 🆕 **New**
   - A tester finds a defect and logs it in the system.
2. 📥 **Assigned**
   - The bug is assigned to a developer for fixing.
3. 🔧 **Open**
   - Developer starts working on the bug.
4. 🔁 **Fixed**
   - Developer fixes the bug and marks it as fixed.
5. □ **Retest**
   - QA retests the issue to confirm the fix.
6. ✅ **Closed**
   - QA confirms the bug is resolved and closes it.

✅ What is User Acceptance Testing (UAT)?

User Acceptance Testing (UAT) is the **final phase of testing** where the **actual users or clients** test the software to ensure that it meets their **business needs and requirements** before it goes live.

🔍 In simple terms: "**Did we build what the customer wants?**"

🎯 Purpose of UAT:

- Verify that the system works **as expected from a real-world perspective**
- Ensure the system is **ready for production**
- Get **final approval** from stakeholders or users

👥 Who performs UAT?

- **End-users**
- **Clients**
- **Business analysts**
- **Product owners**

Not typically the QA team — QA ensures functionality, **UAT ensures business value**

✅ What is Exploratory Testing?

Exploratory Testing is a type of testing in which testers explore the application **without predefined test cases**, relying on their **experience, domain knowledge, and curiosity**.

 It's like: "Let me see how this app behaves if I try this…"

✅ What is Ad-hoc Testing?

Ad-hoc Testing is an **informal, unplanned testing** done **without any documentation, test cases, or strategy**. It's mostly based on the tester's intuition, experience, and understanding of the system.

🔍 It's like: "Let me randomly test this and see if it breaks."

🔧 Key Traits:

- No structure or plan
- Often done once when time is very short
- Focuses on **breaking the system**
- No documentation, assumptions, or notes
- Usually performed **after formal testing**

🔁 Key Differences: Ad-hoc vs Exploratory Testing

Feature	Ad-hoc Testing	Exploratory Testing
Planning	No planning or documentation	Has a goal or mission (charter)
Structure	Completely unstructured	Semi-structured (learn + test + note)
Documentation	Not maintained	Often documented (notes or session reports)
Who Performs It	Anyone (even non-testers sometimes)	Usually skilled testers
When Done	Last-minute or during crunch time	Anytime, especially early in development
Approach	Random, intuitive	Logical, domain-based

✅ How to Handle Unclear Requirements as a QA Tester

Here's a professional and structured approach you can follow:

1️⃣ Ask Questions & Clarify

🔍 Always start with **communication**.

- Reach out to the **Business Analyst (BA)**, **Product Owner**, or **Client**.
- Ask for:
  - Missing user stories or acceptance criteria
  - Sample inputs/outputs
  - Clarification on flows and edge cases

📌 *Example*:
"Can you confirm what should happen when a user enters a wrong OTP 3 times?"

2️⃣ Refer to Available Documents

- Check:
  - Requirement Specifications (SRS, BRD)
  - Wireframes, mockups, UI designs
  - User stories in JIRA or Confluence
  - API documentation (if backend-related)

 Sometimes designs or past releases give you clues.

3️⃣ Collaborate with Developers or Product Team

- Sit with developers to understand the expected behavior
- Sometimes developers implement logic based on client discussion not documented
- Ask:
  "What's the intended output for this module?"

4️⃣ Use Exploratory Testing

- When nothing is clear — explore the application.
- Note down behaviors, assumptions, and log bugs if something looks wrong
- Share your findings with the team for validation

🔍 *Example:* You notice the profile page crashes after editing details — even though it's not documented, this is worth reporting.

5️⃣ Log Assumptions in Test Cases

- If you're forced to proceed, write your test cases with **clear assumptions**:

  📝 *"Assuming that user must enter email in valid format for successful registration."*

This protects you from blame later if the functionality changes.

6️⃣ Push for Requirement Reviews / Refinement

- Raise it during daily stand-ups or sprint planning
- Recommend a **requirement walkthrough** or **demo session** to reduce confusion
- This helps the whole team, not just QA

✅ Summary: What You Should Say in Interviews

🗣️ *"If requirements are not clear, I proactively communicate with stakeholders like the BA or developers to get clarity. I refer to available docs, use exploratory testing if needed, and clearly document assumptions in my test cases. I also encourage requirement refinement sessions to avoid future confusion."*

✅ Challenges Faced While Testing (with Sample Answers)

1️⃣ Unclear or Changing Requirements

Challenge: Requirements were not well-documented or kept changing during the sprint.

What I did:

- Proactively communicated with the BA or developer for clarification.
- Maintained test cases with assumptions noted.
- Adapted test cases quickly when requirements changed.

🗣️ *"During my QA internship, the client frequently changed requirements, especially in the login and dashboard modules. I stayed in close touch with the product team and updated my test cases accordingly."*

2️⃣ Lack of Time for Testing

Challenge: Tight deadlines gave very little time for complete testing.

What I did:

- Prioritized **critical functionality** for testing (Smoke & Sanity).
- Used **exploratory testing** for quick coverage.
- Suggested automation for repetitive flows.

🗣️ *"Before a release in my e-commerce project, I had just a few hours for testing. I focused on payment, cart, and login, while noting lower-risk areas for post-release review."*

3️⃣ Environment or Deployment Issues

Challenge: The test environment was not stable, or builds weren't deployed correctly.

What I did:

- Coordinated closely with the DevOps or developer team.
- Verified builds before starting test cycles.
- Maintained logs of environment-specific issues.

🗣️ *"Sometimes the test server had outdated builds, which wasted time. I made it a habit to verify build versions before executing tests."*

4️⃣ Reproducing Intermittent Bugs

Challenge: Some bugs didn't appear every time (e.g., login failures, random crashes).

What I did:

- Collected logs, screen recordings, and exact steps.
- Tested on different browsers/devices.
- Worked with developers to trace backend logs.

🗣️ *"A session timeout issue only happened on mobile Chrome. I captured a screen recording and helped the dev reproduce it using my steps."*

5️⃣ Cross-Browser & Device Compatibility

Challenge: UI behaved differently on different browsers or screen sizes.

What I did:

- Used BrowserStack for testing across environments.
- Reported issues with screenshots and browser info.
- Suggested responsive design fixes.

☑ What is Agile Methodology?

Agile is a **flexible, iterative** approach to software development where the product is built **incrementally**, and **feedback is gathered continuously** from the customer and stakeholders.

🔁 **Work is delivered in small cycles (called sprints)** instead of one big final release.

🔍 In simple words:
Agile means **building the software in small pieces**, testing each piece, taking feedback, and improving continuously.

☑ How Agile Works (Simplified Flow):

1. **Product Backlog**: List of features, changes, bug fixes (created by Product Owner)
2. **Sprint Planning**: Team selects a small set of items to work on in the next sprint (usually 1–2 weeks)
3. **Sprint Execution**: Team builds and tests features
4. **Daily Stand-ups**: 15-minute team meetings to track progress
5. **Sprint Review**: Demo of the completed work
6. **Sprint Retrospective**: Team discusses what went well and what to improve
7. 🔁 Repeat for next sprint

🎯 Why Agile?

Feature	Benefit
Iterative delivery	Fast and continuous value delivery
Early feedback	Fewer chances of big failures
Flexibility	Easy to handle changing requirements
Collaboration	Developers, QA, and clients work closely

☑ Role of QA in Agile:

As a QA engineer, **you're involved throughout the sprint**, not just at the end!

Your tasks include:

- Participating in **daily stand-ups and planning**
- **Writing and executing test cases** for each user story
- **Performing regression and exploratory testing**
- **Reporting bugs** and ensuring quick resolution
- Contributing to **automation testing** if applicable

✅ What is a Sprint?

A **Sprint** is a fixed time-boxed period (usually **1 to 4 weeks**) in Agile where the team works to **complete a set of tasks or user stories** from the product backlog.

🔁 It's a short, repeatable cycle where **development, testing, and delivery happen together**.

🕒 Most teams use **2-week sprints**.

✅ Summary: What Happens During a Sprint?

Phase	What Happens
Sprint Planning	Stories selected, estimates discussed
Sprint Execution	Dev & QA work on building and testing features
Daily Stand-ups	Team syncs up, blockers discussed
Sprint Review	Demo of finished work
Retrospective	Team reviews process, suggests improvements

✅ How to Write Test Cases for an ATM

An ATM has many common features like:

- Insert Card
- Enter PIN
- Withdraw Cash
- Check Balance
- Transfer Money
- Print Receipt
- Eject Card

You can write test cases using this format:

Test Case ID	Test Scenario	Test Steps	Expected Result
TC_01	Valid Card & Correct PIN	Insert valid card → Enter correct PIN	Login successful → Main menu displayed
TC_02	Valid Card & Wrong PIN	Insert valid card → Enter incorrect PIN 3 times	Card blocked → Show error
TC_03	Expired or Blocked Card	Insert expired or blocked card	Show message: "Invalid card" → Eject card

Test Case ID	Test Scenario	Test Steps	Expected Result
TC_04	Withdraw within available balance	Login → Select Withdraw → Enter amount within balance	Cash dispensed → Receipt optional
TC_05	Withdraw more than available balance	Login → Select Withdraw → Enter amount > balance	Show error: “Insufficient Balance”
TC_06	Invalid amount (like ₹0 or ₹23)	Enter invalid amount	Show error: “Invalid Denomination”
TC_07	Check Balance	Login → Select Balance Inquiry	Display current balance on screen
TC_08	Cancel operation midway	Login → Start withdraw → Press Cancel	Transaction cancelled → Back to home or eject card
TC_09	Card Eject functionality	Insert card → Press Cancel or complete transaction	Card should be ejected properly
TC_10	Power failure during transaction	Mid-transaction, simulate power loss	Transaction rolled back → Balance remains same

---

🔍 Other Scenarios You Can Cover:

✳ Functional:

- Fund transfer between accounts
- PIN change
- Receipt print option working or not

✳ UI/UX:

- Screen clarity in sunlight/night
- Audio prompts (if present)
- Button responses

✳ Negative Testing:

- Enter special characters in PIN
- Insert card upside down
- Multiple cards inserted together (if supported)

✳ Security:

- Session auto logout after inactivity
- No card retained after timeout
- PIN masked with asterisks (***)

☑ How to Test a Search Box

A search box is typically used to search for:

- Products (in e-commerce)
- Posts (in social media)
- Users, articles, files, etc.

🔍 Functional Test Cases

Test Scenario	Expected Result
Enter a valid keyword (e.g., "mobile")	Related results are displayed
Enter an invalid keyword (e.g., "zxcv123")	“No results found” or similar message shown
Press enter key after typing	Search is triggered
Click search button (if available)	Search is triggered
Check case insensitivity ("Mobile" vs "mobile")	Same results for both
Partial text search ("mob" for "mobile")	Auto-suggestions or matching items shown
Clear search field	Results or suggestions disappear

🛑 Negative Test Cases

Test Scenario	Expected Result
Enter blank spaces and search	No results or validation message shown
Enter only special characters (e.g., @#$%)	Handled gracefully, no crash or XSS
SQL Injection (`' OR 1=1 --`)	Application should **not** break or leak data
Very long input (200+ characters)	Should be trimmed or handled properly

---

🎨 UI/UX Test Cases

- Is the search box **visible and aligned** properly?
- Placeholder text like `"Search for products…"` is shown?
- Is the font and border consistent with design?
- Auto-suggestions appear in real-time?
- Search history is shown or not?
- Mobile responsiveness: Can you search properly on small screens?

Performance & Edge Cases

Test Case	Expected Result
Search while offline	Should show "No internet connection"
Search repeatedly (5-10 times quickly)	System shouldn’t crash or lag
Search while typing (instant results)	Results update smoothly
Test across browsers/devices	Search works same on Chrome, Firefox, mobile, etc.

Security Test Cases

- Search query should not be visible in browser history (for sensitive apps)
- Should not accept `<script>` or other XSS attempts

Bonus: Sample Manual Test Case Format

Field	Example
Test Case ID	TC_Search_001
Test Scenario	Verify valid product search functionality
Steps	1. Enter "iPhone" 2. Press Enter
Expected	Results with “iPhone” should be shown
Result	Pass / Fail

Pro Tip (for Interviews):

“I test the search box by covering valid/invalid input, UI alignment, keyboard actions, and edge cases like SQL injection and rapid typing. I also check how results behave under low network and across different devices.”