Follow for more LinkedIn

📘 Section 1: Introduction — What is a Test Case?

Testing is like checking your work before showing it to others.
In software development, a **test case** is a set of **steps written to test a particular function or feature** of an application.

It helps ensure that everything works the way it's supposed to.
A test case acts as a **bridge between requirements and actual behavior** — it tells the tester what to do, what to expect, and what really happened.

Simple Definition:

A **test case** is a written document that describes:

- What to test
- How to test
- What result you expect
- And what result you actually get

If both the **expected result** and the **actual result** match, the test **passes**.
If they don't match, the test **fails** — meaning there might be a **bug** in the system.

🎯 Why Are Test Cases Important?

Imagine you are baking a cake. You follow a recipe:

- Mix flour, sugar, eggs, and milk
- Bake at 180°C for 30 minutes
- Check if the cake is soft and golden

This recipe is just like a **test case** — it gives you the exact steps to get a good result.
If something goes wrong, you can look at your steps and find where the mistake happened.

Similarly, in software testing:

- Test cases **help track mistakes** (bugs).
- They **save time** by providing ready steps for testing.
- They ensure **no feature is missed** during testing.
- They **improve communication** between testers, developers, and managers.

Follow for more LinkedIn

💡 Real-Life Example:

Think about withdrawing money from an ATM.
You insert your card, enter your PIN, select the amount, and expect cash to come out.

If you don't get the money — something went wrong.
So as a tester, you would write a test case like this:

Test Case Example:

- **Step 1:** Insert your ATM card
- **Step 2:** Enter correct PIN
- **Step 3:** Choose "Withdraw" and enter ₹500
- **Expected Result:** ATM should dispense ₹500

If it doesn't — the test fails, and you report it.

That's how test cases help keep systems reliable, just like how recipes ensure the cake tastes right every time.

🧩 Section 2: Steps to Write Test Cases

Writing test cases is one of the most important parts of software testing.

A **good test case** helps testers verify that the software behaves correctly and helps developers understand where something might have gone wrong.

Let's go step-by-step 👇

Step 1: Understand the Requirements

Before writing a single line of a test case, you need to **fully understand what needs to be tested**.
Read the project requirements carefully. Know how the feature should work from the user's point of view.

Example:
If the requirement says *"A user should be able to log in using a valid username and password,"*
you already know that you will write test cases to check:

- Login with correct credentials (positive case)
- Login with wrong credentials (negative case)

Follow for more LinkedIn

- Login with an empty password field

🧠 **Tip:** Think like a user!
Ask yourself — *"What could the user do right?"* and *"What mistakes might the user make?"*

Step 2: Define the Test Case Structure

Every test case should follow a **standard structure** so anyone can read and understand it easily.

Here's a simple structure to follow:

- **Test Case ID:** A unique number or code for the test case. Example: TC001
- **Test Description:** What the test will verify.
- **Pre-Conditions:** Any setup needed before testing.
- **Test Steps:** The step-by-step actions to perform.
- **Test Data:** The data needed for testing.
- **Expected Result:** What should happen ideally.
- **Actual Result:** What really happened during testing.
- **Status:** Pass or Fail based on the result.
- **Remarks:** Any extra notes or comments.

🧠 **Why use a structure?**
Because it makes your work **organized, readable, and repeatable.**

Step 3: Identify Test Scenarios

A **test scenario** describes *what* needs to be tested — it's a high-level idea before you write detailed steps.

For example, for a **Login Page**, your test scenarios might be:

1. Verify login with valid credentials
2. Verify login with invalid credentials
3. Verify login with empty fields
4. Verify login page UI elements (username field, password field, login button, etc.)

Follow for more LinkedIn

Each of these scenarios can later be converted into one or more detailed test cases.

Step 4: Write the Test Cases

Now, it's time to write!
Take each scenario and write down the details in the structure you created earlier.

Example:

Test Case ID: TC001
Test Description: Verify that the user can log in with valid credentials.
Pre-Conditions: User must be registered and know their username and password.
Test Steps:

1. Open the login page.
2. Enter valid username and password.
3. Click the "Login" button.
   Test Data: Username: `testuser`, Password: `password123`
   Expected Result: User should be redirected to the dashboard.
   Actual Result: (Filled after execution)
   Status: Pass/Fail
   Remarks: (Optional notes)

Step 5: Review and Execute

Once the test cases are written, review them carefully before running.
Make sure:

- The steps are clear
- The expected results are realistic
- The data used is correct

After review, start executing them one by one and record the actual results.

Step 6: Update and Maintain

As the software changes, **update your test cases**.
New features may be added, old ones may change, and some might be removed.

Keeping your test cases up to date ensures that testing always matches the product's latest version.

Follow for more LinkedIn

🧠 **In Simple Words:**
Writing test cases is like writing instructions for checking your own homework.
If your steps are clear and easy to follow, others can test the same feature in the same way — even years later!

🧱 Section 3: Components of a Test Case (Detailed Explanation)

Every test case is like a small story — it tells **what to test**, **how to test**, and **what to expect**.
Each part (or "component") of a test case has a specific purpose.
Let's go through them one by one with simple explanations and examples.

1️⃣ Test Case ID

This is a **unique number or code** given to each test case.
It helps to identify and refer to the test case easily.

Example:
`TC001` – Test Case 1 for Login
`TC002` – Test Case 2 for Password Reset

If you have 100+ test cases, these IDs help you track which ones passed or failed.

🧠 **Tip:**
Use a naming pattern like `TC_ModuleName_001` — for example, `TC_Login_001`.

2️⃣ Test Description

This explains **what the test is checking**.
It should be short but clear enough for anyone to understand.

Example:
"Verify that a user can log in with valid credentials."

A good description answers the question: *"What exactly am I testing here?"*

3️⃣ Pre-Conditions

These are the things that **must be true or done before** running the test.

Example:

Follow for more LinkedIn

- The user must already be registered.
- The login page should be accessible.

If pre-conditions are not met, the test result might not be valid.

🧠 **Analogy:**
Before baking a cake, you must have all ingredients ready — flour, sugar, eggs, etc.
Those are your **pre-conditions**!

4️⃣ Test Steps

These are the **actions you perform** during testing.
They must be written clearly and in sequence — just like following a recipe.

Example:

1. Open the login page.
2. Enter the username in the username field.
3. Enter the password in the password field.
4. Click the Login button.

🧠 **Tip:**
Each step should be specific and simple enough that even a new tester can follow it without asking questions.

5️⃣ Test Data

This is the **input** you use for testing.
Test data should include both **valid** and **invalid** values.

Example (in paragraph form):
For testing login functionality, you can use a username like “testuser” and a password like “Password@123.”
You can also try entering a wrong password like “abc123” to check if the system gives an error message.

🧠 **Why It’s Important:**
Without proper test data, you can’t verify if the system behaves correctly under all conditions.

6️⃣ Expected Result

Follow for more LinkedIn

This defines what **should happen** if everything works correctly.
It's the *ideal* output of the test.

Example:
After entering valid username and password, the user should be redirected to the homepage or dashboard.

7️⃣ Actual Result

This is what **really happens** when you execute the test.
Sometimes it matches the expected result — sometimes it doesn't.

Example:
If the user is redirected successfully, the actual result is *"User redirected to homepage successfully."*
If the system shows an error, the actual result will be *"Error: Page not found."*

8️⃣ Status

This tells whether the test **passed** or **failed** based on comparison between the expected and actual result.

Example:

- If Expected = Actual → Status: **Pass**
- If Expected ≠ Actual → Status: **Fail**

🧠 **Tip:**
A "Fail" is not bad news — it helps find a bug before customers do!

9️⃣ Remarks

This field is used to write **additional comments**, such as issues noticed during testing or suggestions for improvement.

Example:
"Login successful, but redirection takes longer than expected (3 seconds delay)."

💡 In Summary:

Each part of a test case has a role to play:

- ID → helps identify

Follow for more LinkedIn

- Description → tells what's being tested
- Pre-condition → sets up the situation
- Steps → guide the tester
- Data → provides input
- Expected & Actual → show results
- Status → marks success or failure
- Remarks → adds extra information

Together, they form a **complete testing blueprint** that ensures your software is checked thoroughly.

🧾 Section 4: Example of a Complete Test Case (Practical Example)

Now that we understand all the parts of a test case, let's bring everything together with a **real-world example**.
Imagine we're testing the **Login Feature** of an e-commerce application called **MyShop**.

🧪 Test Case ID:

TC001 – Verify Login Functionality with Valid Credentials

🧾 Test Description:

To verify that a registered user can successfully log in using the correct username and password.

⚙️ Pre-Conditions:

1. The user must already be registered in the MyShop application.
2. The user should know their correct username and password.
3. The website or application must be up and running.

Follow for more LinkedIn

🧠 **Real-Life Example:**
Before entering your home, you must have the right key. Similarly, before testing login, you must have a valid user account ready.

🪜 Test Steps:

1. Open the browser and navigate to https://www.myshop.com.
2. Click on the **Login** button on the top right corner of the homepage.
3. Enter a valid username in the **Username** field. (Example: `bharath411`)
4. Enter a valid password in the **Password** field. (Example: `Password@123`)
5. Click the **Login** button.
6. Observe the behavior of the system.

🧠 **Tip:**
Always write steps as *actions* (Click, Enter, Select, Verify) — it makes them clear and easy to follow.

🧮 Test Data:

- Username: `bharath411`
- Password: `Password@123`

🧠 **Why Test Data Matters:**
Just like a recipe needs correct ingredients, testing needs correct input data — wrong data leads to wrong results.

🎯 Expected Result:

The user should be successfully logged in and redirected to the **Dashboard/Homepage**.
The username should appear on the top right corner, confirming successful login.

🔍 Actual Result:

(To be filled after execution — this is what the tester actually observes.)
Example: *User successfully redirected to homepage. Username displayed as "Welcome, Bharath."*

Follow for more LinkedIn

📊 Status:

✅ Pass — If user is redirected successfully.
❌ Fail — If login fails or an error message is displayed for valid credentials.

🗒️ Remarks:

- “Login successful, but dashboard took 2 seconds extra to load.”
- “Password field is case-sensitive (works only when exact case is entered).”

🧠 **Why Remarks Are Useful:**
Remarks help in understanding minor issues or observations that don’t cause a failure but might be useful for developers or future testers.

💬 How This Test Case Helps

This single test case might look simple, but it plays a major role in ensuring the most basic functionality — **logging in** — works smoothly.
If users can’t log in, they can’t shop, search, or order products. That means business loss.
So even one small test like this ensures reliability, user trust, and business success.

🧠 Additional Examples (to understand more):

Let’s see two more mini test cases — one positive and one negative.

✅ Test Case ID: TC002 – Verify Login with Invalid Password

Test Description:
Check if the system shows a proper error message when the user enters an incorrect password.

Pre-Conditions:
The user must be registered.

Test Steps:

1. Open login page.
2. Enter valid username (`bharath411`).

3. Enter invalid password (`wrong@123`).
4. Click Login.

Expected Result:
System should display: *"Invalid username or password. Please try again."*
User should **not** be logged in.

Status:
Pass – if error message is displayed correctly.
Fail – if login succeeds with wrong password.

⚠️ Test Case ID: TC003 – Verify Login Without Entering Username and Password

Test Description:
Check the system behavior when user clicks login without filling in any details.

Pre-Conditions:
Login page must be open.

Test Steps:

1. Leave both fields empty.
2. Click Login.

Expected Result:
The system should show a validation message: *"Username and password cannot be empty."*

Status:
Pass – if validation message appears.
Fail – if system accepts empty input.

✨ Key Takeaway:

A test case is not just a document — it's the foundation of quality.
A single well-written test case like "Login Functionality" can prevent multiple customer issues in the future.

🧩 Section 5: Types of Test Cases

Not all tests are the same.
Each type of test case has a specific goal — some check if the application *works right*, while

others check if it *behaves right*.
Let's explore the main types of test cases one by one, with **simple examples** and **daily life comparisons**.

---

1. Functional Test Cases

These test cases check whether a specific feature or function of the application works according to the requirements.
They focus on **"What the system does."**

Example:
You're testing the *Login* feature of an e-commerce website.
Steps: Enter correct username and password → Click Login → Verify homepage loads correctly.

Expected Result:
User should be redirected to the homepage.

In Real Life:
It's like checking if a TV remote changes the channel when you press the button — it's testing the function itself.

Purpose:
✅ To ensure every feature works as designed.

---

2. Non-Functional Test Cases

These test cases check *how* the system performs rather than *what* it does.
It includes performance, usability, reliability, and security testing.

Example:
You test how fast the login page loads after clicking the button.

Expected Result:
The page should load within 3 seconds.

In Real Life:
Imagine you turn on a light switch — the light should glow instantly, not after 10 seconds.
Speed and smoothness matter here.

Purpose:
✅ To ensure the application is fast, secure, and easy to use.

---

3. Positive Test Cases (Happy Path)

These check if the system works correctly when everything is done properly.
It's called the *Happy Path* because it follows the ideal scenario.

Follow for more LinkedIn

Example:
Enter the correct username and password → Click Login → User successfully reaches the dashboard.

In Real Life:
Like inserting the correct ATM card and entering the right PIN — you get your cash without any issue.

Purpose:
✅ To verify the system works as expected under normal conditions.

---

4. Negative Test Cases (Sad Path)

These check how the system behaves when incorrect or unexpected data is entered.
It ensures the system can handle mistakes gracefully without crashing.

Example:
Enter an incorrect password → System should display an error message like *"Invalid password."*

In Real Life:
If you enter the wrong PIN at the ATM, it doesn't give you money — it just warns you politely.

Purpose:
✅ To ensure the system handles invalid inputs safely.

---

5. Boundary Test Cases

Boundary tests focus on *edge values* — the limits of input fields — to see how the application behaves at those boundaries.

Example:
If a password field allows 8–12 characters, test with:

- 7 characters → should fail
- 8 characters → should pass
- 12 characters → should pass
- 13 characters → should fail

In Real Life:
If an elevator can carry up to 8 people, what happens when the 9th person tries to enter?
That's a boundary test!

Purpose:
✅ To check that the system properly handles the minimum and maximum limits.

Follow for more LinkedIn

---

6. Regression Test Cases

Whenever new features are added or bugs are fixed, regression test cases ensure the old functionality still works fine.
In other words — *new changes should not break old features.*

Example:
After adding a “Remember Me” checkbox on the login page, you re-run the old login test cases to make sure login still works normally.

In Real Life:
Imagine you renovated your kitchen. You’d still check if the gas stove works properly afterward, right?

Purpose:
✅ To ensure software stability after updates or changes.

---

7. Smoke Test Cases

Smoke tests are quick checks performed before deep testing starts.
They confirm that the main features are working and that the system is stable enough for detailed testing.

Example:
Before detailed testing, you quickly check:

- Does the login page open?
- Can fields accept input?
- Does the Login button respond?

In Real Life:
Like checking if a car starts before taking it for a long drive. 🚗

Purpose:
✅ To identify major issues early and save time.

---

8. Sanity Test Cases

Sanity tests are done after a small change or bug fix, to ensure the specific functionality now works correctly.

Example:
If a bug in the “Forgot Password” feature was fixed, you just test that particular function again to confirm it’s working.

Follow for more LinkedIn

In Real Life:
If your phone screen was repaired, you just test the touch screen — not the whole phone.

Purpose:
✅ To verify that recent fixes didn't break other parts of the app.

💡 Quick Recap of All Test Case Types

Type	Focus	Example
Functional	What the system does	Login, Search, Add to Cart
Non-Functional	How the system behaves	Speed, Security, Usability
Positive	Correct inputs	Valid Login
Negative	Wrong inputs	Invalid Password
Boundary	Edge values	Password length limits
Regression	Re-testing after change	Old login after new update
Smoke	Quick check	App opens, Login works
Sanity	Focused check	Re-test bug fix

🌟 In Short

Every good tester must know which type of test to perform and when.
Like different doctors for different problems, every test type has its own purpose in keeping software healthy and strong.

Follow for more LinkedIn

🧩 Section 6: Writing Effective Test Cases (Step-by-Step Guide)

Writing a good test case is like writing a clear recipe. It must be **simple, specific, and easy to follow** — so anyone (even a new tester) can understand what to do, what to expect, and how to confirm if the software works correctly.

Let's break this down step by step 👇

🧱 Step 1: Understand the Requirement

Before writing a test case, **read and understand the requirement document or user story** clearly.
For example, if the requirement says:

> "When the user enters valid credentials, they should be redirected to the homepage."

Then your focus should be testing all conditions related to login — valid, invalid, blank inputs, etc.

Tip: Never start writing without clarity. If something is not clear, ask your team or the business analyst.

🗒️ Step 2: Define the Objective

Every test case must have a **clear purpose**. Ask yourself — what are you testing?
Example objective:

> "Verify that the user can successfully log in with valid credentials."

A clear objective ensures your test case has focus and meaning.

🧍 Step 3: Identify Pre-Conditions

Pre-conditions are the things that must be true before the test begins.
Example:

> "User account should already exist in the system."

If preconditions aren't met, the test may fail even if the application works fine.

⚙️ Step 4: Write the Test Steps

Test steps are **actions** performed by the tester. They must be **numbered and written clearly** so anyone can follow them easily.
Example:

Follow for more LinkedIn

1. Open the application.
2. Navigate to the login page.
3. Enter username as `bharath411`.
4. Enter password as `Password@123`.
5. Click on the “Login” button.

Tip: Keep each step to one clear action. Avoid combining two steps into one.

🧪 Step 5: Define Test Data

Mention all necessary inputs.
Example:

> Username: `bharath411`, Password: `Password@123`

If there are multiple data sets (valid/invalid), list them all in the same test case or create separate ones.

🧰 Step 6: Define the Expected Result

This is the **most important part** — what should happen after performing all steps.
Example:

> “User should be successfully redirected to the homepage.”

Expected results help compare the actual result later.

📊 Step 7: Execute the Test

Run the steps on the actual application and observe what happens.
Example:

> After clicking login, user is redirected to the homepage — means it passed.

🗒️ Step 8: Record the Actual Result

Write down what actually happened during execution.
Example:

> “User was successfully redirected to the homepage.”

If there’s any difference from the expected result, it’s a **defect**.

Follow for more LinkedIn

✅ Step 9: Update Status

After execution, mark the test case status as:

- **Pass** – if actual result = expected result.
- **Fail** – if actual result ≠ expected result.
- **Blocked** – if cannot execute due to other issue (e.g., server down).

💬 Step 10: Add Remarks

Remarks help testers note extra observations.
Example:

> "Login working fine on Chrome browser. Need to test on Firefox."

⚠️ Section 7: Common Mistakes in Writing Test Cases (and How to Fix Them)

Even experienced testers can make small mistakes that lead to **confusing, incomplete, or inaccurate** test cases.
Let's go through the most common errors — and learn how to avoid them.

❌ Mistake 1: Writing Vague or Unclear Steps

Some testers write steps like:

> "Login to the application."

That's too short and unclear!
How to fix it ✅

> "Open the application → Click on the Login link → Enter valid username and password → Click the Login button."

Always write steps **clearly and sequentially** so that even a new tester can follow without help.

❌ Mistake 2: Missing Expected Results

If you don't mention what the expected result is, no one can verify whether the test passed or failed.

Follow for more LinkedIn

How to fix ✅
Always write a clear **expected outcome** like:

> “User should be redirected to the homepage after clicking Login.”

Expected results are the **benchmark for validation** — never skip them.

❌ Mistake 3: Combining Multiple Scenarios in One Test Case

Some testers write everything in one test case — login, search, and checkout together.
That makes it hard to track and maintain.
How to fix ✅
Write **one test case per scenario**.
Example:

- TC001 – Login with valid credentials
- TC002 – Login with invalid credentials
- TC003 – Login with blank fields

This helps when running regression tests or reusing cases later.

❌ Mistake 4: Forgetting Preconditions

If you skip preconditions, others might not know the setup required to run your test.
Example mistake:

> “Test checkout process” — but didn’t mention that a user must be logged in.
> How to fix ✅
> Always mention the required setup.
> Example:
> “User must be logged in and have at least one item in the cart.”

❌ Mistake 5: Not Updating Test Cases After Application Changes

Applications often evolve — new features are added, old ones removed.
If you don’t update your test cases, they become outdated.
How to fix ✅
Whenever the software changes, review and update related test cases.
Use **version control or test management tools** like JIRA or TestRail to track updates.

❌ Mistake 6: Not Using Consistent Naming or Formatting

Inconsistent names make your suite messy.
Example:

Follow for more LinkedIn

- TC-1, test_login, LOGIN-TEST — all used for same type of case.
  How to fix it ✅
  Follow a consistent naming format:

  TC001, TC002, TC003 …

It improves readability and helps in automated tracking.

❌ Mistake 7: Ignoring Negative Test Cases

Many testers focus only on positive flows — the happy paths.
But negative testing is equally important.
Example mistake:

Only testing “valid login.”
How to fix it ✅
Add test cases for invalid credentials, blank inputs, and special characters.

Negative cases often uncover real bugs that users might trigger by mistake.

❌ Mistake 8: Missing Post-Conditions

Post-conditions describe the state after execution.
Example mistake:

Forgetting to log out or clean up data.
How to fix it ✅
Add post-conditions like:
“User is logged out and browser closed.”
This ensures a clean test environment for the next run.

❌ Mistake 9: Not Linking Test Cases to Requirements

When test cases are not connected to requirements, coverage gaps appear.
How to fix it ✅
Use **Requirement Traceability Matrix (RTM)** — a simple table showing which test case verifies which requirement.

❌ Mistake 10: Writing Test Cases Without Reviewing

Test cases should always go through **peer review** before execution.
How to fix it ✅
Have another tester or QA lead review your test cases.
Fresh eyes often catch missing data, unclear steps, or formatting issues.

Follow for more LinkedIn

💡 Quick Tip:

> “A good test case is like a clear recipe — anyone can cook the same dish, with the same taste, by following it exactly.”

📝 Conclusion

Test cases are the backbone of software quality. They guide testers step by step, ensure features work correctly, and help catch bugs before users face them. Well-written test cases save time, improve communication between teams, and make software reliable. By following clear structures, using real-life examples, and learning from common mistakes, anyone can write effective test cases that keep applications strong and users happy.

“Every bug you catch and every test case you write is a step toward building software that people trust and enjoy. Your careful testing today makes a better experience for users tomorrow!”