

QA Handbook 01

A Beginner's Guide to Software Testing

Learn software testing from scratch
with simple examples & real QA
practices



Author :

Dewmini Abeywardhana

dewminiabey99@gmail.com

Table Of Content

01.Introduction	
Why This QA Handbook Exists	02
02.Chapter 01	
What is Software Testing?.....	04
03.Chapter 02	
Roles & Responsibilities of a QA Engineer.....	08
04.Chapter 03	
Understanding SDLC & STLC.....	13
05.Chapter 04	
Manual Testing vs. Automation Testing.....	18
06.Chapter 05	
Functional vs. Non-Functional Testing.....	22
07.Chapter 06	
The Bug Life Cycle.....	27
08.Chapter 07	
How to Write a Bug Report.....	31
09.Chapter 08	
How to Write Effective Test Cases.....	35
10.Chapter 09	
QA Tools Every Beginner Should Know.....	41
11.Chapter 10	
Common Mistakes Beginner QAs Make And How to Avoid Them.....	49
12.Your Next Steps in the QA Journey.....	54

Introduction

Why This QA Handbook Exists

Purpose of This Handbook

This handbook is created to help absolute beginners understand the world of software testing.

When I started as a QA, I had many questions like:

- **What exactly is QA?**
- **What does a tester really do?**
- **How do I write test cases or report bugs?**

This guide answers those questions in simple language, with real-life examples and lessons I've learned from experience.

Who Is This For?

- New QA Engineers
- Interns or trainees in software testing
- Students or developers curious about QA
- Anyone who wants to understand how testing works in real projects

How to Use This Handbook

- Go chapter by chapter ,it's designed to be beginner-friendly
- Practice what you learn with your own mini projects
- Use this as a reference while working on real tasks
- Save your favorite tips, templates, and checklists

What Makes This Unique

- Written from a beginner's point of view
- Includes examples from real QA work
- Combines theory + practical tips + templates

What is Software Testing?

Introduction

Imagine you're using a mobile app to send money. You enter the amount, click "Send", and suddenly it crashes. This is exactly what Software Testing aims to prevent.

Software Testing is like the final quality check before users get to use the product. It's where we test whether everything works as expected, and find problems before users do.

Definition (Simple Terms)

Software Testing is the process of checking whether a software product works correctly, meets user requirements, and is free of major bugs or issues.

You don't need to be a developer to test software, you need a user-focused, curious mind that questions things like:

- **What if this button doesn't work?**
- **What happens if I leave this field empty?**
- **Will this app crash if I enter a very large number?**

Purpose of Software Testing

Testing isn't just about finding bugs. It helps ensure:

- The software works as expected
- Errors are caught early
- The product is reliable and user-friendly
- Business goals and user needs are met

Real-Life Analogy

Think of a chef tasting the food before serving it to the customer. That's software testing, making sure the final product is safe, correct, and satisfying.

What Happens Without Testing?

- Apps crash
- Payments fail
- Personal data leaks
- Users lose trust
- The company loses money

One bug in production can cost thousands of dollars or worse, destroy user trust.

Types of Issues Testers Look For

- Functional errors (e.g., login fails)
- UI issues (e.g., text overlaps or missing button)
- Performance problems (e.g., app too slow)
- Security flaws (e.g., sensitive data exposed)

What Makes a Good Tester?

- Think like a user
- Be curious and ask "What if?"
- Pay attention to details
- Communicate clearly (especially when reporting bugs)

Mini Quiz / Self-Check (Optional)

- True or False: Testing is only about finding bugs.
- What would happen if no testing was done on a banking app?
- Name one thing a good tester always does.

Quick Recap

Question	Answer
What is testing?	Checking if the software works correctly
Who does it?	QA Engineers, Testers, sometimes developers too
Why is it important?	To prevent errors, ensure quality, and protect users
Do I need to code?	Not for manual testing, but coding helps in automation

Chapter 02

Roles & Responsibilities of a QA Engineer

Introduction

Being a QA Engineer is more than just "clicking around" and finding bugs.

You are the voice of quality, the last line of defense, and often the first to think like the user. Your role is to make sure the product works well, feels smooth to the user, and doesn't break when it matters most.

Who is a QA Engineer?

A QA (Quality Assurance) Engineer is someone who:

- **Tests software to make sure it works correctly**
- **Finds and reports issues (bugs) before users find them**
- **Works closely with developers, product managers, and designers**
- **Helps maintain product quality through the entire development lifecycle**

You are not just a tester ,you are:

A thinker. A questioner. A quality advocate. A team player.

Key Skills of a Good QA Engineer

- Curiosity – Ask "What if...?" for every scenario
- Detail-Oriented Thinking – Spot even the tiniest mistake
- Communication – Report bugs clearly and respectfully
- User Perspective – Think like a real user would
- Team Collaboration – Work well with Devs, PMs, Designers

Mindset of a QA Engineer

You're not just testing if it works ,You're testing how, when, and what happens when it doesn't.

"It works" is not enough.

"It works in all cases and handles errors gracefully"





That's QA thinking.

QA Involvement Across the Software Development Life Cycle (SDLC)

Phase	QA Role
 Requirements	Review stories, find unclear areas
 Development	Prepare test data and write test cases
 Testing	Execute tests, log bugs, validate fixes
 Deployment	Do smoke testing before go-live
 Post-release	Monitor bugs and gather user feedback

Daily Responsibilities of a QA Engineer

Here's what a typical day might include:

Task	Description
 Review Requirements	Understand what the product is supposed to do
 Write Test Cases	Plan how you'll test the feature
 Execute Tests	Run manual or automated tests on the app/system
 Log Bugs	Report any issues found with details and screenshots
 Re-test Fixed Issues	Check if bugs are fixed properly after dev updates
 Document Results	Maintain records of test results and reports
 Attend Standups/Meetings	Communicate with the team on testing status and blockers

Quick Recap

Point	Summary
What do QAs do?	Plan, execute, and report tests to ensure quality
Key skills?	Curious, detail-focused, good communicator
Do QAs only test?	No — they help improve the entire product experience
Do QAs write code?	Not always (manual), but automation QAs do

Chapter 03

Understanding SDLC & STLC

Introduction

To become a great QA Engineer, you need to understand where you fit in the bigger picture of software development.

That's where two key processes come in:

- **SDLC – Software Development Life Cycle**
- **STLC – Software Testing Life Cycle**

These aren't just buzzwords — they're the maps that guide your team's workflow from idea to delivery.

What is SDLC?

SDLC (Software Development Life Cycle) is the overall process of planning, creating, testing, and deploying software. It shows how software is built, from requirements to release.

What is STLC?







STLC (Software Testing Life Cycle) is the step-by-step process followed by QA teams to test the software effectively. If SDLC is the full movie, STLC is QA's part in that movie.

Real-Life Analogy








Think of building a house:

- SDLC = whole process from designing, building to handing over the keys
- STLC = QA's job, inspecting the house, checking if doors close, taps work, lights turn on

SDLC Phases (Simplified)

Phase	What QA Does
 Requirement Analysis	Understand what to test
 Test Planning	Decide testing strategy, tools, schedule
 Test Case Design	Write test scenarios and cases
 Test Environment Setup	Make sure tools/data are ready
 Test Execution	Run test cases, log bugs
 Test Closure	Final reports, lessons learned, document

STLC Phases (Beginner View)

Phase	What Happens	QA's Role
 Requirement Gathering	Understand what needs to be built	Review requirements, ask questions
 Planning	Plan timelines, resources, tools	Estimate testing effort
 Design	Architects and devs design the system	Think about what can go wrong
 Development	Developers write the code	Prepare test cases, test data
 Testing	QAs test the system	Execute tests, report bugs
 Deployment	Code goes live	Run smoke tests, validate
 Maintenance	Fix bugs, update features	Retest fixes, ensure stability

Why QA Should Know Both

Helps you understand when to start testing

Helps you ask the right questions early

Makes you a smarter team player

Improves your communication with Dev and PM teams

Quick Recap

- SDLC = Full process of building software
- STLC = QA's specific process for testing it
- Knowing both helps you plan, test, and communicate better

SDLC vs. STLC: What's the Difference?

SDLC	STLC
Covers the entire software process	Focuses only on testing
Involves developers, PMs, designers, QA	Involves only QA team
Starts from idea to delivery	Starts once requirements are ready
Ends with product delivery	Ends with test closure/report

Chapter 04

Manual Testing vs. Automation Testing

Introduction

There are two main ways to test software:

- By using your hands and brain
Manual Testing
- By writing scripts to test automatically
Automation Testing

Both are important in real-world projects. But knowing when to use what is the key.

What is Manual Testing?

Manual Testing is when a QA executes test cases without using any automation tools. You check the system yourself by clicking, typing, navigating, and observing results

Example: Manual Testing Flow

Let's say you're testing a login page:

1. **Enter a valid username/password**
2. **Click the Login button**
3. **Check if you're taken to the dashboard**
4. **Log a bug if it doesn't work**

No code. Just your brain, browser, and a test plan.

Pros of Manual Testing

- Great for UI/UX testing
- Better for exploratory testing
- Easy to start with (no coding needed)
- Useful in early project stages

Cons of Manual Testing

- Takes more time to repeat tests
- Human errors possible
- Not ideal for large-scale or repeated testing

What is Automation Testing?

Automation Testing is when you write scripts or use tools to automatically test software — especially repetitive or complex scenarios. Instead of clicking, your script does the work.

Example: Automation Testing Flow

Using Selenium, you can write a script that:

- **Opens the browser**
- **Enters username/password**
- **Clicks Login**
- **Verifies the dashboard loads**
- **Repeats this for 100 different test accounts — in seconds!**

Pros of Automation Testing

- Fast & efficient for repetitive tests
- Great for regression and performance testing
- Increases test coverage
- Saves time in the long run

Cons of Automation Testing

- Requires coding knowledge
- Not all tests can (or should) be automated
- Takes time to build and maintain scripts

Real-Life Analogy

Manual Testing = You check every door and window yourself

Automation Testing = You install sensors that check everything for you

Both help ensure the house (product) is safe, just different approaches.

Quick Recap

Feature	Manual Testing	Automation Testing
Done by	Human	Tools/scripts
Best for	UI, new features, exploratory	Regression, performance
Speed	Slower	Much faster
Skills needed	Observation, logic	Coding, tool knowledge
Use in projects?	Always	When worth automating

Chapter 05

Functional vs. Non-Functional Testing

Introduction

In software testing, we don't just test what a system does, we also test how it behaves.

That's where the two major types of testing come in:

- **Functional Testing** – What the system does
- **Non-Functional Testing** – How well it performs

Both are essential for delivering a quality product.

What is Functional Testing?

Functional testing checks whether the system works as expected based on requirements.

It answers the question: "Does the feature do what it's supposed to do?"

Examples of Functional Testing:

- Can users log in with valid credentials?
- Does clicking "Add to Cart" actually add the item?
- Are error messages shown for invalid inputs?
- Can users reset their password?

What is Non-Functional Testing?









Non-Functional testing checks how well the system performs – like speed, security, and usability.

It answers the question: “Is the product reliable, fast, and user-friendly?”

Examples of Non-Functional Testing:

- How fast does the page load on slow internet?
- Can 500 users use the app at the same time?
- Is the app secure against hackers?
- Is the system easy to use and navigate?

Non-Functional Testing Includes:

Type	Purpose
 Performance Testing	Speed and responsiveness
 Load Testing	How system behaves under expected load
 Stress Testing	How system behaves under extreme load
 Security Testing	Checking for vulnerabilities and data protection
 Compatibility Testing	Across browsers, devices, OS
 Usability Testing	How easy and intuitive the app is
 Recovery Testing	Can it recover from crashes?
 Installation Testing	Is the app easy to install or update?

Real-Life Analogy

Imagine testing a car:

- **Functional Testing:**
 - Does the engine start?
 - Do the brakes work?
 - Do the wipers move?
- **Non-Functional Testing:**
 - How fast does it go from 0–100 km/h?
 - Is the ride smooth?
 - Is the dashboard easy to use?

Key Differences at a Glance

Feature	Functional Testing	Non-Functional Testing
Focus	What it does	How it performs
Based on	Business requirements	Performance & quality attributes
Done by	QA testers	QA + performance/sec
Examples	Login, forms, links	Speed, security, user experience

Quick Recap

- Functional = Does it work?
- Non-Functional = How well does it work?
- Both are equally important for a reliable product

Chapter 06

The Bug Life Cycle

Introduction

As a QA engineer, finding bugs is part of your daily work, but it's not just about spotting them. You need to log them properly, track their status, and communicate clearly until they're fixed.

This journey from discovering a bug to closing it is called the Bug Life Cycle.

Understanding this cycle helps you become a more professional and effective tester.

What Is a Bug?

A bug is any issue where the software behaves differently from what is expected.

If the system does something it shouldn't, or doesn't do what it should, it's a bug.

What Is the Bug Life Cycle?

The Bug Life Cycle (also called Defect Life Cycle) is the process a bug follows from the moment it is found until it is resolved or closed.

It includes different stages and roles you, the developer, sometimes the lead or product manager.

Bug Life Cycle – Key Stages

Let's walk through each step:

Stage	Description
 New	Bug is found and logged by QA
 Assigned	Bug is assigned to a developer
 Open	Developer starts working on the fix
 Fixed	Developer completes the fix
 Retest	QA tests the fix in the new build
 Closed	Bug is fixed and verified – no issue found
 Reopened	QA finds the bug still exists after fix
 Rejected	Developer says it's not a bug or not valid
 Deferred	Fix postponed for future release (low priority or time constraints)
 Duplicate	Similar bug already reported

Example Bug Life Cycle (Login Issue)

1. You find the login button does nothing after clicking
2. You log the bug in JIRA with all details → Status = New
3. Developer is assigned → Status = Assigned
4. Developer fixes it → Status = Fixed
5. You test it again → Status = Retest
6. It works now → You close the bug → Status = Closed

But if it still fails, you change it to → Reopened

Tips for Managing Bugs Professionally

- Write clear bug titles and descriptions
- Always attach screenshots or videos
- Link the bug to relevant test cases or user stories
- Respectfully describe the issue — not blame
- Follow up if status doesn't update

Bug Severity vs. Priority (Bonus Tip)

Term	Meaning	Example
Severity	How bad the bug is from a technical perspective	App crashes = High Severity
Priority	How soon it should be fixed	Login issue = High Priority

You can have a bug that is:

- High severity, low priority (e.g., rare crash)
- Low severity, high priority (e.g., spelling mistake on homepage)

Quick Recap

- A bug follows multiple stages from New → Closed
- QA must track status and verify fixes
- Communication is key between QA and Dev
- Good bug reports save time and reduce confusion

How to Write a Bug Report

Introduction

Finding a bug is only half the job, reporting it clearly is what makes you a great QA engineer.

A well-written bug report saves time, avoids confusion, and helps developers fix issues quickly.

In this chapter, you'll learn:

- **What makes a bug report good**
- **What to include**
- **A simple format you can reuse**

What Is a Bug Report?

A bug report is a formal record of an issue in the software, logged by a tester, so that it can be tracked and fixed by the development team.

Think of it as your communication tool between QA, developers, and stakeholders.

Qualities of a Good Bug Report

- Clear and simple language
- Reproducible steps
- Relevant screenshots or videos
- Specific expected vs. actual results
- No assumptions or emotions

Bad: "Login not working. Pls fix soon!"

Good: "Login button unresponsive after entering valid credentials on Chrome (v123.0) – details below"

Bug Report Format (Simple & Standard)

Here's a format you can follow in JIRA, Excel, or any bug tracker:

Field	Description
Title	Short summary of the issue
Steps to Reproduce	Step-by-step actions to trigger the bug
Expected Result	What should have happened
Actual Result	What actually happened
Severity	Impact level (Low, Medium, High, Critical)
Priority	How urgently it needs fixing
Environment	OS, Browser, App version, Device
Attachments	Screenshots, screen recordings
Status	New, Assigned, Fixed, Closed, etc.
Assigned To	Developer or team responsible

Example Bug Report

Title: Login Button Not Working on Chrome

Steps to Reproduce:

- Navigate to <https://example.com/login>
- Enter valid credentials
- Click the Login button

Expected Result:

User is redirected to the dashboard

Actual Result:

Nothing happens when clicking the Login button

Severity: High

Priority: High

Environment: Windows 10, Chrome v123.0

Attachments: [Screenshot], [Loom video]

Status: New

Assigned To: Frontend Developer – John

Common Mistakes to Avoid

- Vague descriptions (e.g., “page broken”)
- Missing steps or assumptions
- No attachments
- Emotionally written reports
- Not checking if the bug was already reported
-

Tools You Can Use to Log Bugs

- JIRA – Most common in Agile teams
- Bugzilla / GitHub Issues / Trello – Used in various environments
- Excel/Google Sheets – Simple teams or small projects
- Screenshots Tools: Lightshot, Snipping Tool
- Screen Recording: Loom, OBS Studio

Quick Recap

- Bug reports are a key part of QA communication
- Always include steps, results, severity, and evidence
- A clear bug report = faster fixes + better teamwork
- Reuse templates to stay consistent

Chapter 08

How to Write Effective Test Cases

Introduction

A test case is a written set of steps used to verify if a feature in the application works correctly.

Writing clear, meaningful, and reusable test cases is a skill every great QA should master. It's not just about checking functionality, it's about communicating logic in a way anyone can follow.

A good test case tells what to test, how to test, with what data, and what outcome is expected.

-

What Is a Test Case?

A test case includes:

- What to test (the feature)
- How to test it (the steps)
- What input to use (test data)
- What result to expect (expected outcome)

Why Test Cases Matter

- Help testers stay consistent
- Useful for retesting and regression
- Act as a record of coverage
- Can be reused or automated later
- Support teamwork, others can follow your steps easily



Qualities of a Good Test Case

Field	Description
Test Case ID	Unique identifier (e.g., TC_001)
Test Case Title	What are you testing?
Preconditions	Setup or conditions needed before testing
Test Steps	Step-by-step instructions
Test Data	Input data to use
Expected Result	What should happen
Actual Result	What happened (during execution)
Status	Pass / Fail / Blocked
Remarks	Notes, bugs linked, screenshots, etc.

Example Test Case

Test Case ID: TC_001

Title: Test the Submit Exam button

Preconditions:

- User is logged in
- Exam is started

Test Steps:

1. Go to the exam page
2. Answer at least one question
3. Click the Submit Exam button
4. Click Yes on the confirmation popup

Test Data:

- Question 1: Answered (Option A)
- Question 2: Left blank

Expected Result:

- Exam is submitted
- A message appears: "Your exam has been submitted"
- User is redirected to the exam summary or result page

Actual Result:(Write after testing)

Status: Pass / Fail

Positive vs. Negative Test Cases

Type	Example
✓ Positive	Login with valid credentials
✗ Negative	Login with invalid password, blank username, or special characters

Test both – Users don't always do what they're supposed to!

Tips to Improve Your Test Cases

- Keep steps short and precise
- Use action words: Click, Enter, Select, Verify
- Use consistent formatting
- Avoid combining multiple checks in one case
- Review your cases like you're a brand-new tester reading them

Quick Recap

- A test case = steps + data + expected result
- Good test cases = clear, reusable, traceable
- Cover both happy paths and edge cases
- Your test cases reflect your thinking quality as a QA professional

Chapter 09

QA Tools Every Beginner Should Know



Introduction

As a beginner QA, you don't need to know everything — but knowing the right tools will save you time, improve your accuracy, and help you work better with your team.

This chapter introduces the essential tools for different QA tasks — from bug tracking to testing APIs.

-

1. Bug Tracking & Project Management Tools

These tools help you log, assign, and track bugs and manage your test tasks.

Tool	What it's used for
JIRA	Most popular tool for bug tracking and managing tasks in Agile teams
Trello	Simple board-style project tracking (good for small teams or personal projects)
Asana	Task management and tracking with status updates
GitHub Issues	For reporting bugs and enhancements directly on code repositories

2. Test Case Management Tools

These tools help you organize, write, and track test cases.

Tool	What it's used for
TestLink	Open-source tool for managing test cases
TestRail	Popular tool to manage and report test plans (paid)
Google Sheets / Excel	Easy, beginner-friendly way to start writing and sharing

3. API Testing Tools

As a QA, testing APIs is an important skill. These tools let you send requests, check responses, and automate API tests.

Tool	What it's used for
Postman	Beginner-friendly tool for testing REST APIs
Swagger UI	Used to explore and understand API
Insomnia	Alternative to Postman, clean and fast API client

4. Web & UI Testing Tools

For checking front-end behavior, UI issues, and responsiveness.

Tool	What it's used for
Browser Dev Tools	Built into Chrome/Firefox – inspect elements, view console logs, network errors
Selenium	Tool for automating browser-based tests
Lighthouse	Google's tool for testing page performance, accessibility, SEO

5. Automation Testing Tools

These tools help you automate repetitive test cases (usually requires basic coding knowledge).

Tool	Language	Purpose
Selenium WebDriver	Java/Python/C#	Automates browser actions
Playwright	JavaScript/TypeScript	Modern browser automation
Appium	Java/Python	Automates mobile app testing (Android/iOS)
TestNG / JUnit	Java	Used with Selenium for managing test scripts

6. Performance & Security Tools (Basic Awareness)

You don't need to master them yet, but basic knowledge is useful.









Tool	Use
JMeter	Load and performance testing
OWASP ZAP	Basic security testing
Burp Suite	Intercept and analyze traffic (for security testing)

7. Screenshot & Recording Tools

Use these tools to capture bugs or steps for easier reporting.

Tool	Purpose
Lightshot / Snipping Tool	Take quick screenshots
Loom / OBS Studio	Record screen with or without voice for bug reports

Quick Recap

Task	Tool Examples
 Bug Tracking	JIRA, Trello, GitHub Issues
 Test Case Writing	TestLink, TestRail, Google Sheets
 API Testing	Postman, Swagger UI
 UI Testing	Chrome DevTools, Selenium
 Automation	Selenium, Appium, Playwright
 Performance	JMeter
 Security	OWASP ZAP
 Screenshots	Lightshot, Loom

Chapter 10

Common Mistakes Beginner QAs Make And How to Avoid Them

Introduction

Nobody starts as a perfect QA. We all learn by doing and making mistakes.

This chapter lists real mistakes many beginner QA engineers make and shows you how to fix them early, so you can grow faster and become more confident in your role.

If you've made some of these ,don't worry, you're not alone!

-

1. Writing Vague Bug Reports

The Mistake:

"Login not working. Please fix."

Why it's bad:

It gives no context. Developers waste time guessing what went wrong, where, and how.

How to fix it:

Write clear, step-by-step bug reports:

- What page?
- What inputs?
- What did you expect?
- What happened instead?

Example:

"On the Login page, after entering valid credentials and clicking Login, nothing happens. Expected: Redirect to dashboard."

2. Skipping Negative Test Scenarios

The Mistake:

Only testing the "happy path"—where everything works correctly.

Why it's bad:

Real users make mistakes. Systems must handle invalid, missing, or incorrect data.

How to fix it:

Always test both positive and negative cases:

- Empty fields
- Wrong input formats
- Special characters
- Invalid user behavior

Example:

Try logging in with a wrong password, blank email, or script input.

3. Not Reproducing Bugs Before Logging

The Mistake:

Logging a bug without confirming it happens every time.

Why it's bad:

Leads to “Can’t reproduce” responses from developers.

How to fix it:

Try it 2–3 times first. Check if it’s consistent or happens under specific conditions.

Bonus tip:

Always mention the browser, device, or API data used.

4. Poor Test Case Documentation

The Mistake:

Writing messy, unclear, or incomplete test cases.

Why it's bad:

Other testers can’t follow your steps. You’ll forget what you tested later.

How to fix it:

Use a clean, simple format. Include:

- Title
- Preconditions
- Test steps
- Test data
- Expected result
- Status

Make it easy enough that someone new can follow and test confidently.

5. Not Asking Questions Early

The Mistake:

Staying silent when requirements are unclear or test flows feel confusing.

Why it's bad:

Leads to guessing, wasted time, or missing key test cases.

How to fix it:

Speak up early. Ask the developer or PM:

- "What should happen when the timer ends?"
- "Should this button be visible for all users?"

6. Ignoring Edge Cases

The Mistake:

Only testing with perfect, standard data.

Why it's bad:

You'll miss bugs that only appear under strange or extreme input.

How to fix it:

Think like a creative user:

- What if the field has 255 characters?
- What if the form is submitted twice quickly?
- What if I switch tabs while taking the exam?

7. Forgetting to Retest and Close Bugs

The Mistake:

Bug is fixed, but you don't go back to test it again.

Why it's bad:

Bugs remain open or get marked incorrectly.

How to fix it:

Keep a bug checklist. Retest fixes regularly and update the bug status (e.g., "Verified," "Closed").

8. Not Keeping Notes

The Mistake:

Testing without recording results or observations.

Why it's bad:

You won't remember what passed, what failed, or what you skipped.

How to fix it:

Maintain a simple daily test log or notes:

- What you tested
- Bugs found
- Unusual behavior
- Pending items

Bonus Tip: Don't Be Too Hard on Yourself

- Making mistakes is part of learning.
- The best QAs learn from feedback, fix things fast, and keep improving.

You've Finished the QA Handbook
What's Next?

Your Next Steps in the QA Journey

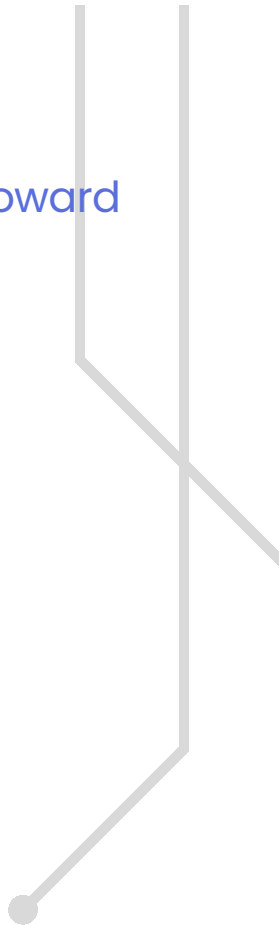


Congratulations! You've just taken your first step toward becoming a strong, thoughtful QA Engineer.

You now understand:

- **What QA really means**
- **How to write test cases and bug reports**
- **Which tools to start with**
- **How to avoid beginner mistakes**
- **And how to think like a real tester**

But your journey doesn't stop here.



“Great QA isn’t about just finding bugs — it’s about thinking critically, caring about users, and improving every day.”

You don’t need to be perfect — just stay curious, ask questions, and keep learning.

Thank you for reading. You’ve got this. ❤️

**-Dewmini Abeywardhana -
QA Engineer**

Connect With Me

I believe in sharing what I learn.

You can follow my content and growth here:

Medium | GitHub



Dewmini Abeywardhana
dewminiabey99@gmail.com