Quality Assurance & Testing Standards

26 June 2024

Published by
Montrose Software
montrosesoftware.com

Table of Contents

1. Types of testing

You may perform part or all of the following testing activities during your work:

1. Accessibility Testing
Accessibility testing is the practice of ensuring your mobile and web apps are working and usable for users without and with disabilities such as vision impairment, hearing disabilities, and other physical or cognitive conditions.

2. Acceptance Testing
Acceptance testing ensures that the end-user (customers) can achieve the goals set in the business requirements, which determines whether the software is acceptable for delivery or not. It is also known as user acceptance testing (UAT).

3. Black Box Testing
Black box testing involves testing against a system where the code and paths are invisible.

4. White Box Testing
White box testing involves testing the product's underlying structure, architecture, and code to validate input-output flow and enhance design, usability, and security.

5. Functional Testing
Functional testing checks an application, website, or system to ensure it’s doing exactly what it’s supposed to be doing.

6. Interactive Testing
Also known as manual testing, interactive testing enables testers to create and facilitate manual tests for those who do not use automation and collect results from external tests.

7. Integration Testing
Integration testing ensures that an entire, integrated system meets a set of requirements. It is performed in an integrated hardware and software environment to ensure that the entire system functions properly.

8. Load Testing
This type of non-functional software testing process determines how the software application behaves while being accessed by multiple users simultaneously.

9. Non-Functional Testing
Non-functional testing verifies the readiness of a system according to nonfunctional parameters (performance, accessibility, UX, etc.) which are never addressed by functional testing.

10. Performance Testing
Performance testing examines the speed, stability, reliability, scalability, and resource usage of a software application under a specified workload.

11. Regression Testing
Software regression testing is performed to determine if code modifications break an application or consume resources.

12. Sanity Testing
Performed after bug fixes, sanity testing determines that the bugs are fixed and that no further issues are introduced to these changes.

13. Security Testing
Security testing unveils the vulnerabilities of the system to ensure that the software system and application are free from any threats or risks. These tests aim to find any potential flaws and weaknesses in the software system that could lead to a loss of data, revenue, or reputation per employees or outsides of a company.

14. Single User Performance Testing
Single user performance testing checks that the application under test performs fine according to specified threshold without any system load. This benchmark can be then used to define a realistic threshold when the system is under load.

15. Smoke Testing
This type of software testing validates the stability of a software application, it is performed on the initial software build to ensure that the critical functions of the program are working.

16. Stress Testing
Stress testing is a software testing activity that tests beyond normal operational capacity to test the results.

17. Unit Testing
Unit testing is the process of checking small pieces of code to ensure that the individual parts of a program work properly on their own, speeding up testing strategies and reducing wasted tests.

18. End-to-End Testing
End-to-End testing is a technique that tests the application’s workflow from beginning to end to make sure everything functions as expected.

2. Traceability Matrix

A traceability matrix is a document that details the technical requirements for a given test scenario and its current state. It helps the testing team understand the level of testing that is done for a given product.

The traceability process itself is used to review the test cases that were defined for any requirement. It helps users identify which requirements produced the most number of defects during a testing cycle.

Not only does this show areas in need of improvement, but it also helps mitigate future roadblocks and identify process weaknesses.

A requirements traceability matrix (RTM) is a tool that helps identify and maintain the status of the project's requirements and deliverables. It does so by establishing a thread for each component. It also manages the overall project requirements.

There are many kinds of RTMs. For example, a test matrix is used to prove that tests were conducted. It can also be used to identify issues and requirements during the development of software.

An RTM ensures that projects do everything they set out to do. This step-by-step process helps identify the requirements and the products that are required to be tested successfully. It also helps in determining the project's direction and timeline.

First, it will support the identification of all requirements in a work product. Then, it will check to make sure there is coverage of all the requirements throughout the project's lifetime.

The RTM will show the requirements coverage in terms of the number of test cases, design status, and execution status. It will also show the UAT status for a specific test case. With all this information at your fingertips, your team will be able to analyze changes in requirements and make informed product development decisions on the fly.

And because traceability links artifacts across the development lifecycle, it helps teams identify and resolve issues before they become problems. It can also help avoid the pressure of an audit. And if you do get audited, having an RTM will make it easier to demonstrate that you have complied with regulations which means you can avoid additional expenses or delays the audit may cause.

You can even use it to track requirements from compliance regulations in a compliance matrix. That will help you understand what you need to test and develop before the work is finalized.

In a nutshell: a requirements traceability matrix makes it easier to meet goals and manage projects.

What do you include in a requirements traceability matrix?

Create a simple chart with the following columns:

- Requirements: Add sub-columns for marketing requirements, product requirements, and system-level specifications (if applicable).
- Testing: Add a sub-column for test cases and test runs.
- Deviation: Add a sub-column for any issues.

Example of traceability matrix

Requirements Traceability Matrix

	Testing								Defects			
WBS Deliverables	Test Case ID	Test Description	TEST	UAT	QA	PROD	PRE-PROD	NON-PROD	Defective	Defect ID	Defect Description	Req. Status
WBS-001	TC001	Verify user can successfully register an account	Pass	Pass	Pass	Pass	Fail		No			Complete
WBS-001	TC002	Verify error message is displayed for invalid inputs	N/A	Fail	Pass	Fail	N/A		Yes	DEF001	Invalid email format	Complete
WBS-001	TC003	Verify user receives a confirmation email	Pass	N/A	Fail	N/A	Pass		Yes	DEF002	Email not sent	In Progress
WBS-002	TC004	Verify user can log in with valid credentials	Fail	Pass	N/A	N/A	N/A		NO			In Progress
WBS-002	TC005	Verify error message is displayed for incorrect login	Fail	Fail	N/A	Pass	Fail		Yes	DEF003	Incorrect username	Complete
WBS-002	TC006	Verify "Forgot Password" link redirects correctly	Fail	Fail	N/A	Pass	Pass		No			In Progress
WBS-003	TC007	Verify user can create a new post				N/A	N/A		No			Not Started
WBS-003	TC008	Verify error message is displayed for empty content	Pass	Pass	Pass	Fail	Fail		Yes	DEF004	Empty post content	Complete

3. Test Scenario

Test Scenarios are created for the following reasons:

- Creating Test Scenarios ensures complete Test Coverage.
- Test Scenarios can be approved by various stakeholders like Business Analyst. Developers, Customers to ensure the Application Under Test is thoroughly tested. It ensures that the software is working for the most common use cases.
- They serve as a quick tool to determine the testing work effort and accordingly create a proposal for the client or organize the workforce.
- They help determine the most important end-to-end transactions or the real use of the software applications.
- For studying the end-to-end functioning of the program, Test Scenario is critical.

Test Scenarios may not be created when:

- The Application Under Test is complicated, unstable and there is a time crunch in the project.
- Projects that follow Agile Methodology like Scrum, Kanban may not create Test Scenarios.
- Test Scenario may not be created for a new bug fix or Regression Testing. In such cases, Test Scenarios must be already heavily documented in the previous test cycles. This is especially true for Maintenance projects.

How to Write Test Scenarios

As a tester, you can follow these five steps to create Test Scenarios:

Step 1: Read the Requirement Documents like BRS, SRS, FRS, of the System Under Test (SUT). You could also refer to use cases, books, manuals, etc. of the application to be tested.

Step 2: For each requirement, figure out possible users actions and objectives. Determine the technical aspects of the requirement. Ascertain possible scenarios of system abuse and evaluate users with a hacker's mindset.

Step 3: After reading the Requirements Document and doing your due Analysis, list out different test scenarios that verify each feature of the software.

Step 4: Once you have listed all possible Test Scenarios, a Traceability Matrix is created to verify that each & every requirement has a corresponding Test Scenario.

Step 5: The scenarios created can be reviewed by your supervisor, and later by other Stakeholders in the project.

Tips to Create Test Scenarios

1. Each Test Scenario should be tied to a minimum of one Requirement or User Story as per the Project Methodology.

2. Before creating a Test Scenario that verifies multiple Requirements at once, ensure you have a Test Scenario that checks that requirement in isolation.

3. Avoid creating overly complicated Test Scenarios spanning multiple Requirements.

4. The number of scenarios may be large, and it is expensive to run them all. Based on customer priorities only run selected Test Scenarios.

4. Test Case

Types of test cases

Test cases can be categorized based on the purpose they serve in testing. As a quality assurance professional, knowing the difference between them helps focus your efforts and choose the right test format.

Functionality test cases: These are the most basic and obvious test cases to write. They ensure that each feature of your system works correctly.

Performance test case: This test ensures that the system runs fast enough. It makes sure that all system requirements work as expected regarding speed, scalability, or stability.

Unit test cases: Software developers usually write unit tests for their code to check individual units, for example, modules, procedures, or functions, to determine if they work as expected.

User interface (UI) test cases: It's important to remember that the user interface is part of the overall system and not just a shell where functionality appears. UI test cases check that each UI element works correctly, displays, and is easy to use.

Security test cases: Security test cases help ensure that a product or system functions properly under all conditions, including when malicious users attempt to gain unauthorized access or damage the system. These test cases safeguard the security, privacy, and confidentiality of data.

Integration test cases: These ensure that the application components work together as expected. These test cases check whether modules or components integrate seamlessly to form a complete product.

Database test cases: These test cases ensure that the database meets its functional and non-functional requirements. They make sure database management systems (DBMS) support all business requirements.

Usability test cases: Usability test cases help check if users can use the application successfully. These determine whether users can easily use the system without difficulty or confusion. They also verify if users can navigate the system using common procedures and functions.

User acceptance test cases: User acceptance test cases verify that an application satisfies its business requirements before users accept it. These determine whether users accept or reject the output produced by a particular system before release to the live environment.

Regression testing: Regression test cases verify that changes made during development don’t cause any existing functionality to stop working. Regression testing happens after changes have been made to existing code to test that all existing or legacy functionality continues to work as expected after making the changes.

How to create a test case?

Test cases are the blueprints that testers will follow, so they must be clear, thorough, and accurate. Below, we've outlined 10 steps you can take whether you're writing new test cases or revisiting and evaluating existing test cases.

1. Define the area you want to cover from the test scenario.
2. Ensure the test case is easy for testers to understand and execute.
3. Understand and apply relevant test designs.
4. Use a unique test case ID.
5. Use the requirements traceability matrix in testing for visibility.
6. Include a clear description in each test.
7. Add proper preconditions and postconditions.
8. Specify the exact expected result.
9. Utilize suitable testing techniques.
10. Get your test plan peer-reviewed before moving forward.

Make test cases reusable and maintainable wherever possible. Your needs will vary depending on the software, application, or specific features you're testing. However, you can save time and energy by consciously creating test cases that are reusable and easy to maintain.

Create test cases with the end user's perspective in mind. Remember throughout the test case writing process that you're trying to step into the user's place. Aligning your exploratory testing methods with the user's perspective will help you create efficient and relevant software application test cases.

Here are a few elements you can add to your test case template:

- Test Case ID
- Test Case Description
- Pre-Conditions
- Test Steps
- Test Data
- Expected Result
- Post Conditions
- Actual Result
- Status

5. Test Scenario vs Test Case

- A Test Case is a set of actions executed to verify particular features or functionality, whereas a Test Scenario is any functionality that can be tested.
- Test Case is mostly derived from test scenarios, while Test Scenarios are derived from test artifacts like BRS and SRS.
- Test Case helps in exhaustive testing of an application, whereas Test Scenario helps in an agile way of testing the end-to-end functionality.
- Test Cases are focused on what to test and how to test, while Test Scenario is more focused on what to test.
- Test Cases are low-level actions, whereas Test Scenarios are high-level actions.
- Test Case requires more resources and time for test execution, while Test Scenario requires fewer resources and time for test execution.
- Test Case includes test steps, data, and expected results for testing, whereas Test Scenario includes an end-to-end functionality to be tested.

6. Error reporting instructions for testers

It aims to standardize the method of error reporting, which will make the cooperation between
developers and testers easier and more effective. Its purpose is to make work easier for both
parties.

Error reporting steps:

1. When you spot a bug try to repeat it.
2. Make sure you are using the correct version of the app/web.
3. Make sure this does not meet the requirements (that this is really a bug).
4. Find the pattern and conditions for the bug.
5. Check if this bug has already been reported.
6. Report the bug if it was not yet reported.

How to Write Bug Report:

There is no exact bug report template, as it depends upon your bug-tracking system. Your template might be different.

However, the following common fields are always needed when you want to write a bug report:

- Bug ID/Title.
- Severity and Priority.
- Description
- Environment
- Steps to reproduce.
- Expected result.
- Actual result.
- Attachments (screenshots, videos, text, logs)

1) Title/Bug ID:

Every bug should be given a unique identification number. Bug reporting tools should be unique numbers for the newly raised bugs so we can easily identify the bug.

Examples:

❌ Bad: “I can’t see the product when I type again, it doesn’t appear.”

Vague
Aggressive
Too wordy
asks for a solution to be implemented.

✅ Good: “CART – New items added to the cart that do not appear”.

This kind of Title instantly locates the issue (CART)
It focuses on the actual technical problem.

2) Bug Severity:

Bug severity is a very important factor in the bug report. It describes the effect of the defect on the application’s performance.

Blocker: This error causes the app to fail.
Major: A critical error indicates a major change in the business logic.
Minor: An issue that doesn’t affect the application’s functionality but does affect the expected results.
Trivial: It does not affect the functionality or operation of the app. It could be a typographical error.

3) Bug Priority:

Following is the general gradation to decide bug priority:

High: It covers anything that affects the flow or blocks app usage.
Medium: It adversely affects the user experience.
Minor: All other errors like (typos, missing icons, layout issues, etc.).

4) Environment:

A Bug can appear in a specific environment and not others. For example, sometimes a bug appears when running the website on Firefox, or an app malfunctions only when running on an Android device and working fine on an iPhone.

These bug reports can only be identified with cross-browser or cross-device testing. So, when reporting the bug, QAs should be able to specify if the bug should be observed in one or more specific environments.

5) Summary:

However, adding only the Title in the bug report does not serve the purpose. So, if your Title isn't enough, you can add a short report summary.

Your summary in as few words as possible including when and how the bug occurred. Your Title and bug description should also be used in searches, so you must ensure you have covered important keywords.

Examples:

Bad: "I was trying to add stuff to the test, and nothing showed up when I did that or clicked on the button."
Good: "When I tried adding [PRODUCT] to the shopping cart, but nothing happened when I clicked the 'add' button on the specific product overview webpage."

6) Steps to reproduce:

When reporting a bug, it is important to specify the steps to reproduce it. You should also include actions that may cause the bug. Here, don't make any generic statements.

Be specific on the steps to follow:

Here, is an example of a well-written procedure:

Steps:

1. Select product X1.
2. Click on Add to cart.
3. Click Remove to remove the product from the cart.

7) Expected result:

In bug reports, describing the expected result per the technical task, test case outcomes design, or according to the tester's opinion is important. All this helps developers to focus on quickly finding needed information.

For example:

Required fields should be highlighted in red after clicking the" Submit" button.

8) Actual result:

As its name suggests, this field describes the actual effect of the bug. It is very important to write a clear description of the actual result.

For example:

Required fields are highlighted in green color after clicking the "Submit" button.

9) Attachments (screenshots and videos):

In bug reports, it is best practice to attach files to bug reports which makes it easier to perceive information when you need to display it visually:

For example:

Screenshot: Screenshots can easily elaborate mistakes in the program; s convenient when the bug is highlighted with a specific annotation, circle, or arrow image).
Video: Sometimes, it is difficult to describe the bug in words, so it's better to create a video so that the developer can rectify the defect in the program).

7. Contact information

We are happy to address any questions you may have about our services or proposal. Please feel free to reach out to any of us at any time.

Colin Strasser, CEO
colin.strasser@montrosesoftware.com

Marek Krzynówek, Head of Business Development
marek.krzynowek@montrosesoftware.com

Marek Koprowski, Growth Manager
marek.koprowski@montrosesoftware.com

Our Offices

USA
351 Hartford Rd
South Orange NJ 07079

Poland
Twardowskiego 65
30-346 Kraków