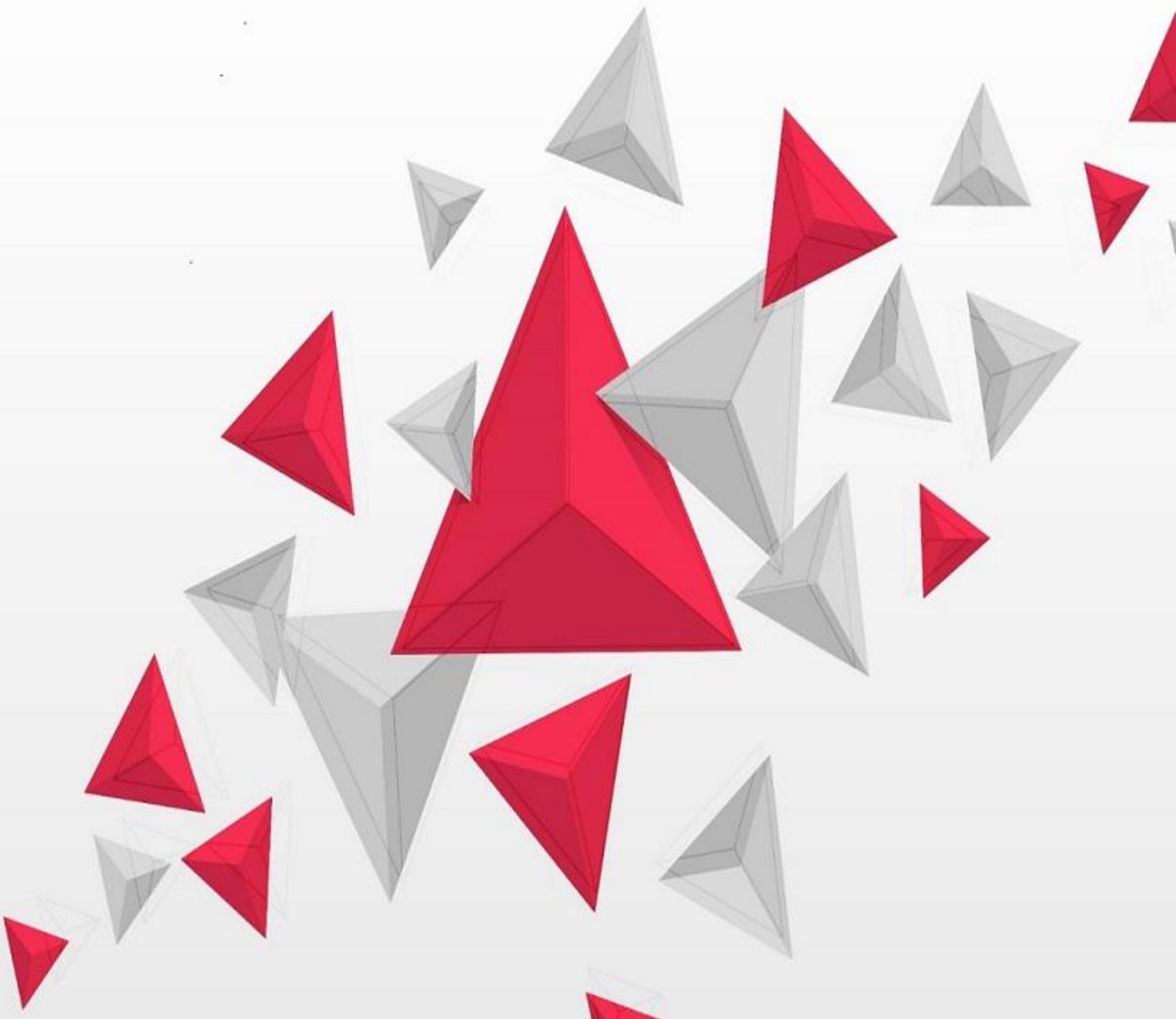


# Test Case Guide Book





## TEST CASE – INTRODUCTION

Now you might be wondering why I have added the test case section under this course as test cases are the responsibility of a quality control guy, and a Business Analyst doesn't have much to do with it. Right?

Wrong!

Since a well-tested work product is a critical success factor for any project, seasoned BA should not only be aware of the basics of testing but should be able to create a test case document when asked for!

In some of the BA job descriptions, you will find the ability to execute acceptance test cases as one of the job responsibilities. Besides, in the process of creating detailed test cases, analysts discover scenarios or flows that were missed during the requirement gathering and documentation.

So now, let's get down to business.

*The rationale behind having a test case document is clearly defining the specifications required to evaluate whether a particular project feature is working as specified in the project requirement document.* Test cases contain a set of test data, preconditions, steps to execute those conditions, and the expected or ideal results that one should get after executing those conditions.

On the onset, let me clarify the difference between a **Test Case** and a **Test Scenario** – the former describes a single step to achieve a specific task while the latter is a sequence of test cases executed one after the other to achieve a functionality or a scenario. Thus, writing your user id on the login page is a test case, and the whole process of logging in to an application is a test scenario.



## ASPECTS OF A TEST CASE

The structure of this particular lesson is a bit different from others, and rather than going through the characters and benefits of a test case, we will spend time learning about the testing basics essential for a business analyst.

### Test plan

A test plan is prepared in the planning stages of a project, and much like a Requirement Management Plan, it contains the scope of the testing for the project i.e.

- What all functionalities need to be tested
- The testing activities to be carried
- Their schedule
- Associated documentation
- How the issues or bugs are to be reported

The test plan is prepared by the testing lead of the project and is approved & baselined by the PM.

### Unit Testing

Unit testing is also called **module-level testing**, and it covers the testing of only a single module or functionality.

Unit testing is almost always done by the developer and on his machine/system. To ensure that the functionalities are developed the right way, the BA should oversee the unit test cases prepared by the developers and sometimes perform some unit testing herself.

## Integration Testing

The testing performed when two or more individual functionalities or modules are integrated as a group is called Integration or String testing. Integration testing happens after the unit testing and ensures the modules' data communication flow and integrity.

The developers usually perform integration testing; however, based on organization processes or project needs, the testers or the business analysts can also be called in to validate the functionality before it goes for the system testing.

## System Testing

Post a successful integration test; the complete application is tested for functionality, business cases, performance, and end-user experience under the System Testing. This kind of testing is the primary responsibility of the project's QC (Quality Control) team, and the test cases are used as the base documents to conduct system testing.

System testing is also called **black-box testing** as the internal implementation or code is neither known nor tested by the tester [*for the curious ones out there, **white box testing** is one in which the underlying functional code and logical conditions are validated and tested*].

## User Acceptance Testing (UAT)

The application testing carried out by the end-users or the client to validate whether the system is performing as documented under the baselined requirements is called User Acceptance Testing or beta testing. This class of testing is typically conducted in a production-like environment and with a valid set of data.

Owing to Business analysts' complete knowledge of the application and the acceptance criteria, they play a vital role in the UAT. Usually, they are the ones who demonstrate the product to the client, followed by the client and their team playing with the system themselves.

The UAT decides whether the quality of the application is acceptable to the client and determines whether the project is a 'Go,' i.e., fit for being moved to a production environment or 'No-Go,' i.e., some functionalities are not working as expected and needs to be discussed and fixed.

## Regression Testing

Software Applications are regularly subjected to modifications - be it functionality enhancements, integration with other applications, or security updates, and it's essential to validate that the old functionalities should not break while incorporating new ones.

Regression testing does precisely that and makes sure that the old code should work as it used to, and any new changes should neither break existing functionality nor introduce new issues.

Regression testing is the responsibility of the testing/QC team.

## Load Testing

Most of the applications being built today are internet-based and can be accessed by multiple users simultaneously. Thus, the boundaries of the application need to be defined by testing the maximum number of users (i.e., the 'Load') that could simultaneously use the application without having any considerable impact on the application performance.

Automated load testing tools usually carry out this kind of testing like Apica load tester or Apache JMeter.

Great! So now, since you are equipped with the basic knowledge of the testing process and types in a typical project, we can go ahead and learn how to create a test case document.



## HOW TO CREATE A TEST CASE

As a pre-requisite, you should have the functionality matrix and the respective use case/user story document with you against the test case you are attempting to create. Discussed ahead are the individual fields that form a test case document and details what goes inside each of those fields.

### Basic Details

This section contains the necessary details about the test case document:

- **Project Name:** Name of the project for which the test case document is being created
- **Module Name:** Name of the module (section) for which the test cases are being written
- **Release version:** The number/version of the release under which the test case's respective functionality will be delivered
- **Test designed by:** Name of the associated user who is creating the test case document
- **Test designed date:** Date on which the test case document is being created

### Test Case Details

This section contains the particulars of all the test cases included in the document:

- **Test Case ID:** Here, the ID of the specific test case should be mentioned, which is usually in the format of the project code followed by a numerical identifier like 'ATP-003'.
- **Test Case Title:** Here goes the title of the specific test case, which should not be more than 8-10 words long.

- **User Story/Use Case reference:** Since every test case should be traceable to an equivalent requirement document, the name of the specific document should come here
- **Priority:** The priority of the requirements in terms of their importance to the project's success. So, it should be 'High' or 'Medium' for test cases belonging to business rules and major functionalities and 'Low' for other less important ones.
- **Pre-conditions:** Here come any of the conditions that must be fulfilled before executing the test case. Ideally, this field should not be left blank as it gives any new tester the perspective around how they need to test the case.
- **Test Data:** The details of the data that acts as an input for this test case go under this section. You might find it somewhat time-consuming to write the test data the first time, but since this data can be re-used while executing the test case, it will compensate for the time spent.
- **Test Steps:** The order-wise steps of executing the test case should be written here. Other details like testing browser, testing role, and test device can also be added here.
- **Expected Results:** The details of the desired system output after executing the test case should be written here. It should include redirection flow details, success/failure messages, and any alerts that are displayed on the screen.

### Test case execution summary

The 'Test case execution summary' section is updated when a tester tests and executes the cases listed in the test case document. Since tests are repeatedly performed and executed during the lifetime of a project, this section must be labeled like Summary – Sprint 1, Summary – Sprint 2, etc... with more sections added to the right.

- **Test executed by:** To have the project knowledge divided equally amongst the team, different testers will test the same functionality over time. The name of the test case executioner should come here.
- **Test execution date:** The date on which the test case was executed will be mentioned here.
- **Status:** The status of the test execution, i.e., whether the case passed/failed/not executed, should come in here
- **Actual results:** The system's actual result after executing the case should be listed here. While filling this section, the tester should furnish enough details to describe the system behavior.

- **Defect IDs:** If the test case fails, the details of the defects raised against the test case should come here. These details will come in handy when the next time this functionality is being tested.
- **Tester Notes:** Any additional notes that the tester would like to add should come under this column.





## TEST CASE - BEST PRACTICES

1. Make sure your test cases are concise; they only furnish the necessary details and aren't accidentally describing the functionality in the test steps or expected results sections.  
Your aim should be that your test cases are easy to understand and executable for other testers.
2. While creating a test case, think from the perspective of an end-user who is working on the application. As a tester, one should have the complete functional knowledge to be able to write meaningful test cases, and that's why we, the business analysts, are called in for validating the functional tests.
3. Put in efforts to make your test case re-usable and modular by writing generic and not application-specific statements. While this may not be possible for all the test cases, you can still author a fair percentage of the test cases without being application-specific.  
Such exercise aids in re-using the test cases across projects and saves a lot of time *and* effort. So, the next time you find yourself creating some test cases, it may be worthwhile to check if there are test cases already written for a similar or a generic functionality!
4. Test cases are essential documents that assure the quality of any project or product and shouldn't be left unreviewed. If a senior reviewer is unavailable, then a peer review (i.e., an examination by the fellow analysts), senior developers, technical lead, or anybody else who knows the functionality will help.  
It is always advisable to run your documents through an extra pair of eyes.

5. The **test coverage** should be maximum possible - a term you may frequently hear in the project lifecycle.

**Test Coverage** means that all the requirements mentioned in the functionality matrix and/or the use case/user story document should have equivalent test cases. The QC team should strive to achieve 100 percent coverage of the project scope.

6. The test cases, just like the other project artifacts, should be kept updated with the latest changes, and obsolete test cases should be removed. However, it should not be that difficult.

If you, as the project's BA, are doing your job of keeping your RTM up to date, then identifying the test cases that need to be updated should be relatively straightforward.