Test Plan vs Test Strategy – Difference Between Them

Key Difference Between Test Plan and Test Strategy

- Test Plan is a document that describes the scope, objective and weight on software testing task whereas Test Strategy describes how testing needs to be done.
- Test Plan is used at the project level whereas Test Strategy is used at the organization level.
- Test Plan has the primary goal of how to test, when to test and who will verify whereas Test Strategy has the primary goal of what technique to follow and which module to check.
- Test Plan can be changed whereas Test Strategy can't change.
- Test Plan is carried out by the test manager whereas the Test Strategy is carried out by the project manager.

In this Test Plan vs Test Strategy tutorial, you will learn:

- What is Test Plan?
- Test Strategy
- Difference Between Test Strategy and Test Plan

What is Test Plan?

A Test Plan is defined as a document which outlines the scope, objective, method and weight on a software testing task

Test Strategy

Test Strategy in software testing is defined as a set of guiding principles that determines the test design & regulates how the software testing process will be done. The objective of the Test Strategy is to provide a systematic approach to the software testing process in order to ensure the quality, traceability, reliability and better planning.

Test Plan V/s Test Strategy is a prominent confusion among multiple levels of QA Aspirants

Below is the detailed guide to it

Difference Between Test Strategy and Test Plan

Test Plan	Test Strategy
A test plan for software project can be defined as a document that defines the scope, objective, approach and emphasis on a software testing effort	Test strategy is a set of guidelines that explains test design and determines how testing needs to be done
Components of Test plan include- Test plan id, features to be tested, test techniques, testing tasks,	Components of Test strategy includes- objectives and scope, documentation formats,

features pass or fail criteria, test deliverables, responsibilities, and schedule, etc.	test processes, team reporting structure, client communication strategy, etc.
Test plan is carried out by a testing manager or lead that describes how to test, when to test, who will test and what to test	A test strategy is carried out by the project manager. It says what type of technique to follow and which module to test
Test plan narrates about the specification	Test strategy narrates about the general approaches
Test plan can change	Test strategy cannot be changed
Test planning is done to determine possible issues and dependencies in order to identify the risks.	It is a long-term plan of action. You can abstract information that is not project specific and put it into test approach
A test plan exists individually	In smaller project, test strategy is often found as a section of a test plan
It is defined at project level	It is set at organization level and can be used by multiple projects

When to Stop Testing (Exit Criteria in Software Testing)

March 20, 2023

Exit criteria in Testing:

“Well begun is half done” – Applies everywhere, even software testing.

Often we see software testers very enthusiastic at the beginning of the project. We create testing documents such as Test Strategy, Test Plan or Test Cases eagerly and enthusiastically.

Then we get to testing software with a BANG! This is only amplified by the interesting defects we find at the beginning of the project. Getting them resolved will only add to our accomplishment.

As we find loads of defects and complete the first run we move on to the next phase. When we get to the second run we kind of relax and as is the general human tendency of getting bored with testing the same thing in the second run.

Many testers feel that it becomes monotonous work in later runs and start losing interest in testing the same software over and over again. When we reach to, maybe the third run, one question starts haunting us and that is "When to Stop Testing the software?"

I bet you must have felt the same way and asked, "When to stop testing?", at least once. I would say the question is **"When, where and how to stop Testing?"** :)

Conceptually we have read and many testers believe that there cannot be a specific condition or equation to decide "When to stop testing?" There are a number of factors to consider before we conclude on this question.

In today's article, I would like to share my thoughts on how to conclude testing activities when we reach to a point in our testing cycle where we can say this testing is enough. We will do this with the help of a few real life examples in a typical testing cycle.

What You Will Learn: [hide]

- When is it Enough Testing?
- Stopping when all defects are found: Is it possible?
- Decision to stop testing: Exit criteria
- What is Completion or Exit Criteria?
- What should be present in Exit Criteria?
- Testing can be stopped when:
- Conclusion:
- Recommended Reading

When is it Enough Testing?

When can we say that this much testing is enough? Can testing ever be completed?

In order to answer these questions, we will have to analyze testing activities from start to end. Please note that – I am going to define these activities in a lay man’s terms – Not in a fancy technical way.

Let’s consider you are beginning testing of a new project.

Initial Activities:

- The testing team receives requirements.
- The testing team starts planning and designing.
- Initial Test documents are ready and reviewed.

Testing Run #1)

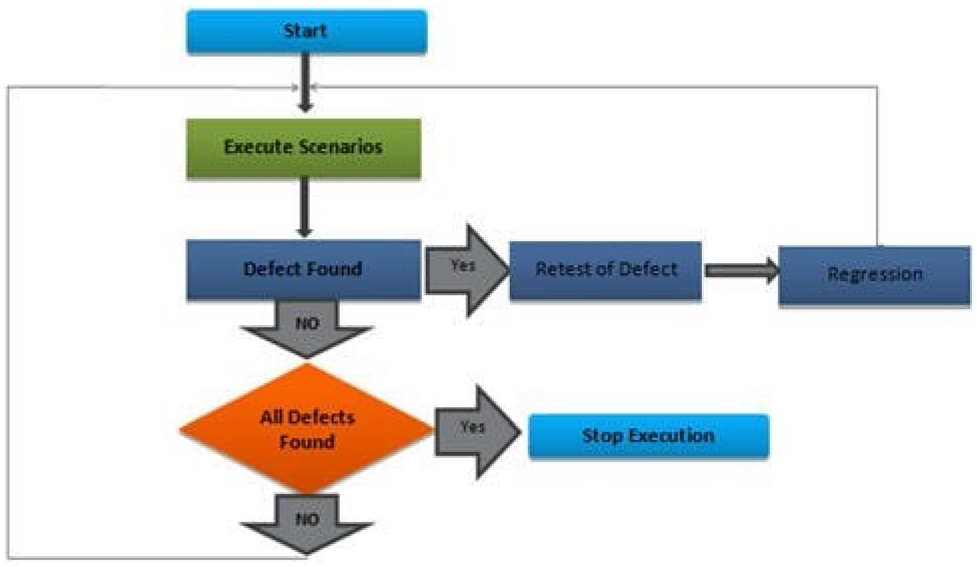
- The testing team starts test execution once they receive the developed product.
- During the testing phase, they execute various scenarios in order to break the software and find many defects. (Also, the defect rate here is higher because the application is new and is undergoing evaluation for the very first time.)
- The Defects get fixed by developers and returned back to test team for retest.
- The testing team performs retesting of the defects and executes regression.
- Once most of the high severity defects are resolved and the software looks stable, development team releases the next version.

Testing Run #2)

- The testing team starts the second run of testing and similar activities are executed as Run 1.
- In this process during the second testing run, few more defects are caught.
- The Defects get fixed by developers and returned back to the test team for a retest.
- Testing team re-tests the defects and performs regression.

This can continue forever. Run 3, Run 4 onwards until all defects in the software are found and software becomes bug-free.

If we want to draw a flow chart for these activities, it will look roughly like below:



From the above flow chart, we can clearly conclude that testing can continue until all defects in the software are found.

But the question is – Is it possible to find every single defect in the software? Let's try to find the answer for this million dollar question :).

Stopping when all defects are found: Is it possible?

Most software is complex and has an enormous testing scope. It is not impossible to find all defects in the software but it will take forever.

Even after finding many bugs in the software, **no one can actually guarantee that the software is defect free now.** There cannot be a situation where we can confidently say that we have completed testing, found all defects in the software and it does not have any more bugs.

Moreover, the purpose of testing is not to find each and every defect in the software. The intent of software testing is to prove that the software does work as intended by breaking it or finding deviation between its current behavior and expected behavior.

There are unlimited defects in software and hence it's impractical to test it until all defects are found as we can never know which defect is the last one. The truth is we cannot depend on finding all the defects in the software to conclude our testing.

Honestly speaking, testing is endless and testing cycles will continue until a decision is made when and where to stop. Now it becomes even more complicated to come to a decision to stop testing. If "stopping when all defects are found" is not the criterion to stop testing then on what basis should it be decided?

Decision to stop testing: Exit criteria

Let's now try to understand – What are the most important factors to be considered while concluding testing activities? I feel the decision to stop testing is mostly dependent on **Time, Budget and Extent of Testing.**

The most common approach is to stop when either Time / Budget is exhausted or all test scenarios are executed. However, with this approach, we will be compromising on the quality of testing and this will not give enough confidence about the software; how?

Let's see with an example.

Test Scenario:

Suppose you are testing a software module. You have been allocated certain budget to cover it. The project timelines are for a month. Total Test Scenarios are 200. You are the only one testing this module.

Scenario #1)

Week 1: You find the showstopper / severity 1 defect on day 1 and the entire testing is blocked for 3 days. Hence you are not able to execute any of the scenarios until Severity 1 defect is resolved. After losing 3 days, the blocker is resolved and you continue with your execution.

At the end of the week, you complete 20 scenarios with few more important high priority defects.

Week 2: You start testing the software putting more focus on defects finding. You open few more Severity 1, Severity 2 and Severity 3 defects during the second week and at the end of the week, you are able to cover 70 Scenarios.

Week 3: By the start of the 3rd week you get all the high priority defects resolved so along with execution of pending scenarios you now have to re-test all the defects which have landed in the testing bucket. Continuing with the good progress you cover 120 scenarios with additional defects.

By this time all high priority defects are already found and reported. So, now you have only Severity 3 defects left to be found.

Week 4: By week 4 you need to re-test most of the opened defects and remaining 80 Scenarios. With this by the end of week 4, you are able to complete up to 180 scenarios with all High and Medium priority defects fixed and re-tested.

Putting this information in Tabular form:

Weeks	Test Activities Performed	Result at the end of the Week
Week 1	• Day 1 - Show Stopper Defect Found. • Testing is Blocked due to Severity 1 defect found on Day 1. • Blocker defect resolved on Day 4. • Test Execution continued until end of week 1.	• High Priority / Critical Defects opened. • 20 Scenarios completed.
Week 2	• More focus on defects finding. • Execution of remaining Test Scenarios. • Re-testing of fixed defects.	• Few more Severity 1, Severity 2 and Severity 3 defects opened. • Total cover 70 Scenarios covered.

Weeks	Test Activities Performed	Result at the end of the Week
Week 3	• Re-testing of all high priority defects. • Execution of Remaining Test Scenarios. • Only Severity 3 defects left to be found.	• Few more Severity 1, Severity 2 and Severity 3 defects opened. • Total cover 120 Scenarios covered.
Week 4	• Re-Testing of all High and Medium Priority Defects. • Execution of remaining Test Scenarios.	• Few more Severity 3 defects opened. • Total cover 180 Scenarios covered.

Should you stop here?

The reason that you have exhausted **Testing Time completely** and have reported and fixed most of the high priority defects. Will stopping at this point give you confidence about the software? Not really due to below reasons:

- Scenarios are not executed completely.
- Few flows are not even tested once.
- All the Scenarios which are covered are executed only once.
- Software still has defects in it.
- Regression is not covered.

Scenario #2)

Week 1: You find Severity 1 defect on day 1 and complete testing is blocked for 3 days. Hence you are not able to execute any of the scenarios until Severity 1 Defect is resolved. After losing 3 days the blocker is resolved and you continue with your execution.

At the end of the week, you complete 20 scenarios with few more defects. This week remains same as Scenario 1.

Week 2: You open few more Severity 1, Severity 2 and Severity 3 defects during the second week, but the focus is to cover more scenarios to cover backlog from week 1. At the end of the week, you are able to cover 120 Scenarios.

Week 3: By the start of the 3rd week you get all the open defects resolved so along with execution of pending scenarios you now have to re-test all the defects which are landed in a testing bucket. Still continuing with good progress at the end the no of scenarios completed becomes 200 with additional defects.

Now you can only report Severity 2 and Severity 3 defects.

Putting this information in Tabular form:

Weeks	Test Activities Performed	Result at the end of the Week
Week 1	• Day 1 - Show Stopper Defect found. • Testing is Blocked due to Severity 1 defect found on Day 1. • Blocker Defect Resolved on Day 4. • Test Execution continued until End of week 1.	• High Priority / Critical Defects opened. • 20 Scenarios completed.
Week 2	• Focus is on executing more scenarios in order	• Few more Severity 1, Severity 2

Weeks	Test Activities Performed	Result at the end of the Week
	to cover for the Backlog from previous week. • Re-testing of Fixed defects.	and Severity 3 defects Opened. • Total cover 120 Scenarios covered.
Week 3	• Re-testing of All high priority defects. • Execution of Remaining Test Scenarios. • Only Severity 3 and few Severity 2 defects left to be found.	• Few more Severity 1, Severity 2 and Severity 3 defects Opened. • All Scenarios covered.

Should you stop here?

You have **covered all Testing scenarios completely** once and have opened few major defects. Will stopping at this point give you confidence about the software?

Not really due to below reasons:

- All Scenarios are executed only once.
- Software still has many major defects in it.
- Regression is not covered.

We can see that the quality is compromised in above both the scenarios. The best approach will be finding a point where all the factors from scenario 1 and scenario 2 are covered and quality is also not compromised. To achieve this we will have to define certain criteria at the beginning of testing.

These criteria are termed as Completion or Exit Criteria. **It is the answer to our question** – "When to stop Testing?".

What is Completion or Exit Criteria?

The exit criteria get evaluated at the end of the testing cycle and is defined in Test Plan. It is the set of conditions or activities which must be fulfilled in order to conclude testing.

The Exit criteria define how much testing is enough and when testing activities can be declared complete. Coverage and completion criteria are combined to define exit criteria for testing.

What should be present in Exit Criteria?

Ideally, Exit or Stop Criteria is defined by combining various factors and hence is unique across all projects. It depends on the project requirement and hence should be defined during Test Planning; at the beginning of the project. Parameters defined in it should be quantified as much as possible.

Below are few pointers to be considered while defining Exit Criteria in case of Functional or System Testing. You may combine few or all the below factors while deciding where to stop testing as per your project needs.

Testing can be stopped when:

Requirements:

- 100% Requirements coverage is achieved.

Defects:

- Defined / Desired Defect count is reached.
- All Show Stopper defects or Blockers are fixed and No known Critical / Severity 1 defect is in Open Status.
- All High Priority defects are identified and fixed.
- Defect Rate falls below defined acceptable rate.
- Very few Medium Priority defects are open and have a workaround in place.
- Very few low priority open defects that do not impact software usage.
- All High Priority defects are re-tested and closed and corresponding Regression scenarios are successfully executed.

Test Coverage:

- Test Coverage should be 95% achieved.
- Test case Pass Rate should be 95%. This can be calculated by formula
    - (Total No of TCs Passed / Total number of TCs) * 100.
- All critical Test cases are passed.
- 5% Test cases can be failed but the Failed Test cases are of low priority.
- Complete Functional Coverage is achieved.
- All major functional / business flows are executed successfully with various inputs and are working fine.

Deadlines:

- Project Deadline or Test Finish deadline is reached.

Test Documents:

- All Test Documents / deliverables (Example – Test Summary Report) are prepared, reviewed and published across.

Budget:

- Complete Testing Budget is exhausted.

“Go / No Go” Meetings:

- “Go / No Go” meeting has been conducted with stakeholders and a decision is made whether the project should go to production or not.

Conclusion:

Let’s make it very simple at the end.

Please answer questions with a simple Yes or No.

If you get most of the answers as Yes that means you can stop testing at this point. If most of the answers are No that means you must check what is missing from testing and this may help you find an escaping production defect :)

- Are all test cases executed at least once?
- Is the Test Case Pass rate as defined?
- Is complete test coverage achieved?
- Are all functional / Business flows executed at least once?
- Is the decided defect count reached?
- Are all Major High Priority Defects fixed and closed?
- Have all Defects been Retested and closed?
- Has Regression been done for all open defects?
- Have you exhausted the testing budget?
- Has the Testing end time reached?
- Are all Test Deliverables reviewed and published?

Software Testing – Requirement Traceability Matrix

Here we will discuss the Requirement Traceability Matrix (RTM). The following 8 topics will be discussed:

1. **What is RTM?**
2. **Parameters To Be Included In RTM**
3. **Why Is RTM Required?**
4. **Who Needs RTM?**
5. **Types Of Traceability Matrix.**
6. **How To Create RTM?**
7. **Advantages Of RTM.**
8. **RTM Template.**

What is Traceability Matrix (TM)?

A Traceability Matrix is a document that co-relates any two-baseline documents that require a many-to-many relationship to check the completeness of the relationship.

It is used to track the requirements and to check the current project requirements are met.

What is Requirement Traceability Matrix?

Requirement Traceability Matrix (RTM) is a document that maps and traces user requirement with test cases. It captures all requirements proposed by the client and requirement traceability in a single document, delivered at the conclusion of the Software development life cycle. The main purpose of Requirement Traceability Matrix is to validate that all requirements are checked via test cases such that no functionality is unchecked during Software testing.

RTM stands for **Requirement Traceability matrix**. RTM maps all the requirements with the test cases. By using this document one can verify test cases cover all functionality of the application as per the requirements of the customer.

- **Requirements:** Requirements of a particular project from the client.
- **Traceability:** The ability to trace the tests.
- **Matrix:** The data which can be stored in rows and columns form.

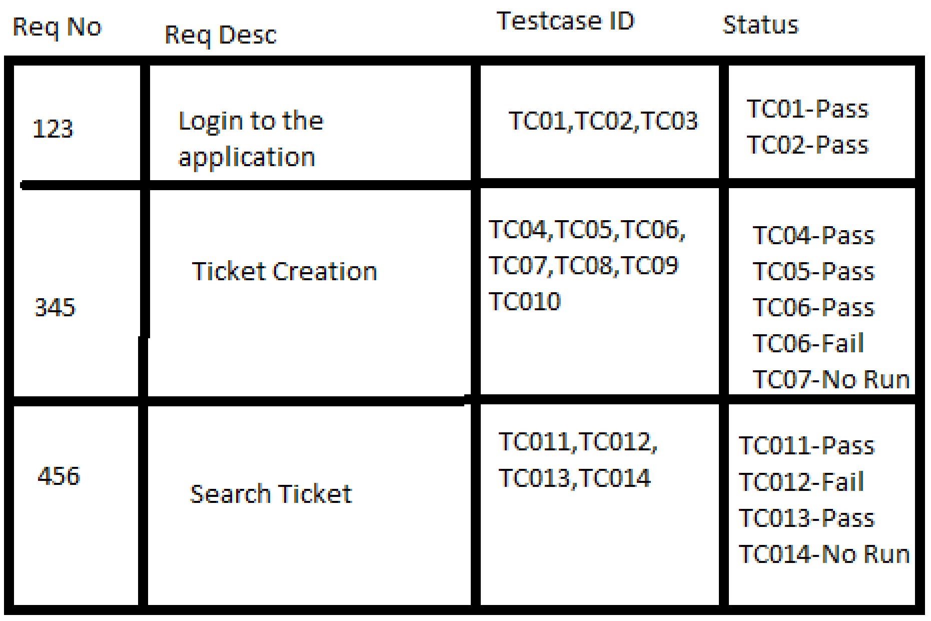
The main purpose of the requirement traceability matrix is to verify that the all requirements of clients are covered in the test cases designed by the testers. In simple words, one can say it is a pen and pencil approach i.e. to analyze the two data information but here just we are using an excel sheet to verify the data in a requirement traceability matrix.

Which Parameters to include in Requirement Traceability Matrix?

The following are the parameters to be included in RTM:

1. **Requirement ID:** The requirement id is assigned to every requirement of the project.
2. **Requirement description:** for every requirement a detailed description is given in SRS (System/Software Requirement Specification) document.
3. **Requirement Type:** understand the type of requirements i.e. banking, telecom, healthcare, traveling, e-commerce, education, etc.
4. **Test cases ID:** Test cases are designed by the testing team. Test cases are also assigned with some ID.

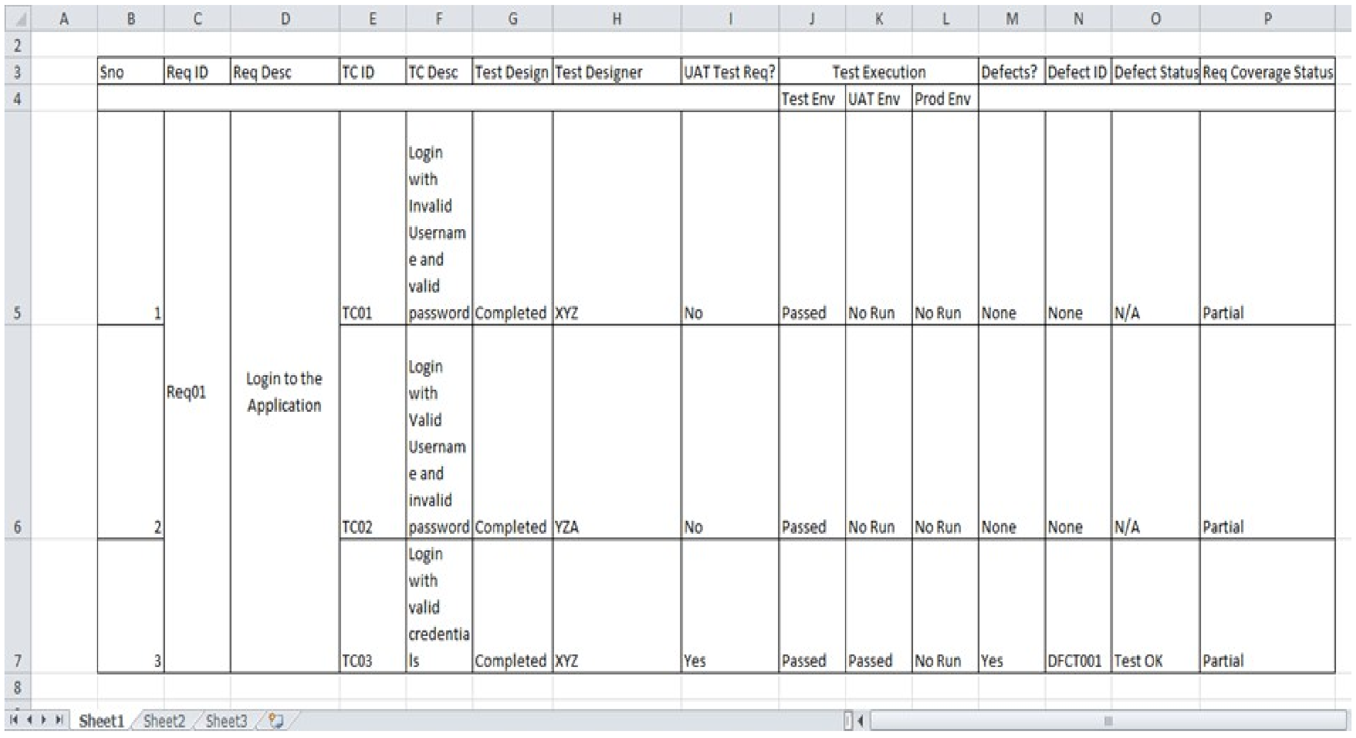
- Requirement ID
- Requirement Type
- Description
- Test Cases ID
- Status of Test case (Pass/Fail)



Req No	Req Desc	Testcase ID	Status
123	Login to the application	TC01,TC02,TC03	TC01-Pass TC02-Pass
345	Ticket Creation	TC04,TC05,TC06, TC07,TC08,TC09 TC010	TC04-Pass TC05-Pass TC06-Pass TC06-Fail TC07-No Run
456	Search Ticket	TC011,TC012, TC013,TC014	TC011-Pass TC012-Fail TC013-Pass TC014-No Run

Above is a sample requirement traceability matrix.

But in a typical software testing project, the traceability matrix would have more than these parameters.



Sno	Req ID	Req Desc	TC ID	TC Desc	Test Design	Test Designer	UAT Test Req?	Test Execution			Defects?	Defect ID	Defect Status	Req Coverage Status
								Test Env	UAT Env	Prod Env				
1	Req01	Login to the Application	TC01	Login with Invalid Username and valid password	Completed	XYZ	No	Passed	No Run	No Run	None	None	N/A	Partial
2			TC02	Login with Valid Username and invalid password	Completed	YZA	No	Passed	No Run	No Run	None	None	N/A	Partial
3			TC03	Login with valid credentials	Completed	XYZ	Yes	Passed	Passed	No Run	Yes	DFCT001	Test OK	Partial

Sheet1 Sheet2 Sheet3

As illustrated above, a requirement traceability matrix can:

- Show the requirement coverage in the number of test cases
- Design status as well as execution status for the specific test case
- If there is any User Acceptance test to be done by the users, then UAT status can also be captured in the same matrix.
- The related defects and the current state can also be mentioned in the same matrix.

This kind of matrix would be providing **One Stop Shop** for all the testing activities.

Apart from maintaining an excel separately. A testing team can also opt for requirements tracing available Test Management Tools.

Why is RTM Important?

When business analysis peoples get the requirements from clients, they prepare a document called SRS (System/Software Requirement Specification) and these requirements are stored in this document. If we are working in the Agile model, we called this document Sprint Backlog, and requirements are present in it in the form of user stories.

When QA get the SRS/Sprint backlog document they first try to understand the requirements thoroughly and then start writing test cases and reviewing them with the entire project team. But sometimes it may happen that in these test cases some functionality of requirements is missing, so to avoid it we required a requirement traceability matrix.

- Each test case is traced back to each requirement in the RTM. Therefore, there is less chance of missing any requirement in testing, and can 100% test coverage can be achieved.
- RTM helps users to discover any change that was made to the requirements as well as the origin of the requirement.
- Using RTM, requirements can be traced to determine a particular group or person that wanted that requirement and it can be used to prioritize the requirement.
- It helps to keep a check between requirements and other development artifacts like technical and other requirements.
- The Traceability matrix can help the tester identify whether by adding any requirement previous requirements are affected or not.
- RTM helps in evaluating the effect on the QA team to reuse the test case.

--OR________________________________

The main agenda of every tester should be to understand the client's requirement and make sure that the output product should be defect-free. To achieve this goal, every QA should understand the requirement thoroughly and create positive and negative test cases.

This would mean that the software requirements provided by the client have to be further split into different scenarios and further to test cases. Each of this case has to be executed individually.

A question arises here on how to make sure that the requirement is tested considering all possible scenarios/cases? How to ensure that any requirement is not left out of the testing cycle?

A simple way is to trace the requirement with its corresponding test scenarios and test cases. This merely is termed as 'Requirement Traceability Matrix.'

The traceability matrix is typically a worksheet that contains the requirements with its all possible test scenarios and cases and their current state, i.e. if they have been passed or failed. This would help the testing team to understand the level of testing activities done for the specific product.

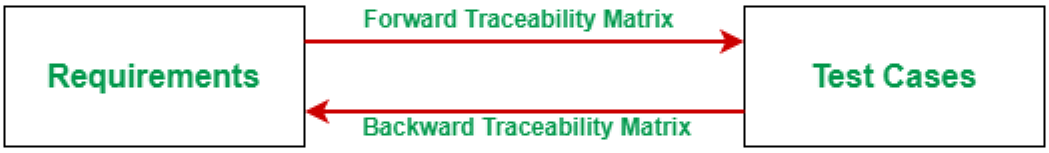
Who Needs RTM?

When testers design the test cases they need to check whether test cases cover all functionality of the application as per the requirements of the customer given in the SRS/Sprint backlog.

- To verify that they need a requirement traceability matrix.
- They generally use an excel sheet or google spreadsheet for RTM.

Types of Traceability Matrix

There are 3 types of traceability matrix:



1. Forward traceability matrix: In the forward traceability matrix, we mapped the requirements with the test cases. Here we can verify that all requirements are covered in test cases and no functionality is missing in test cases. It helps you to ensure that all the requirements available in SRS/ Sprint backlog can be traced back to test cases designed by the testers. It is used to check whether the project progresses in the right direction.

In forward traceability matrix:

Rows = Requirement ID

Column = Test case ID

2. Backward traceability matrix: In the backward traceability matrix, we mapped the test cases with the requirements. Here we can verify that no extra test case is added which is not required as per our requirements. It helps you to ensure that any test cases that you have designed can be traced back to the requirements or user stories and you are not extending the scope of the work by just creating additional test cases that can not be mapped to the requirement. The backward traceability matrix is also known as the **reverse traceability matrix**.

In backward traceability matrix:

Rows = Test cases ID

Column = Requirement ID

3. Bi-directional traceability matrix: A bi-directional traceability matrix is a combination of a forward traceability matrix and a backward traceability matrix. Here we verify the requirements and test cases in both ways.

Bi-directional traceability matrix = Forward traceability matrix + Backward traceability matrix

--OR--

In Software Engineering, traceability matrix can be divided into three major components as mentioned below:

- **Forward traceability**: This matrix is used to check whether the project progresses in the desired direction and for the right product. It makes sure that each requirement is applied to the product and that each requirement is tested thoroughly. It maps requirements to test cases.
- **Backward or reverse traceability:** It is used to ensure whether the current product remains on the right track. The purpose behind this type of traceability is to verify that we are not expanding the scope of the project by adding code, design elements, test or other work that is not specified in the requirements. It maps test cases to requirements.
- **Bi-directional traceability (Forward+Backward):** This traceability matrix ensures that all requirements are covered by test cases. It analyzes the impact of a change in requirements affected by the Defect in a work product and vice versa.

How To Create RTM?

Before creating RTM SRS/Sprint backlog documents and test cases documents are required. Below are the steps to create RTM:

1. For RTM we will use an excel sheet.
2. Write the name of the project, date, and name of the person who is responsible for RTM.
3. Write all requirement IDs row-wise in the first column of an excel sheet.
4. Write all the requirement descriptions row-wise in the second column of an excel sheet.
5. Write all the requirements type row-wise in the third column of an excel sheet.
6. Write all the test cases with their IDs column-wise in an excel sheet.
7. After writing all requirements and test cases you have to verify that for every requirement you have prepared the test cases in both positive and negative flow.

Advantages of RTM

Below are some of the benefits of using RTM:

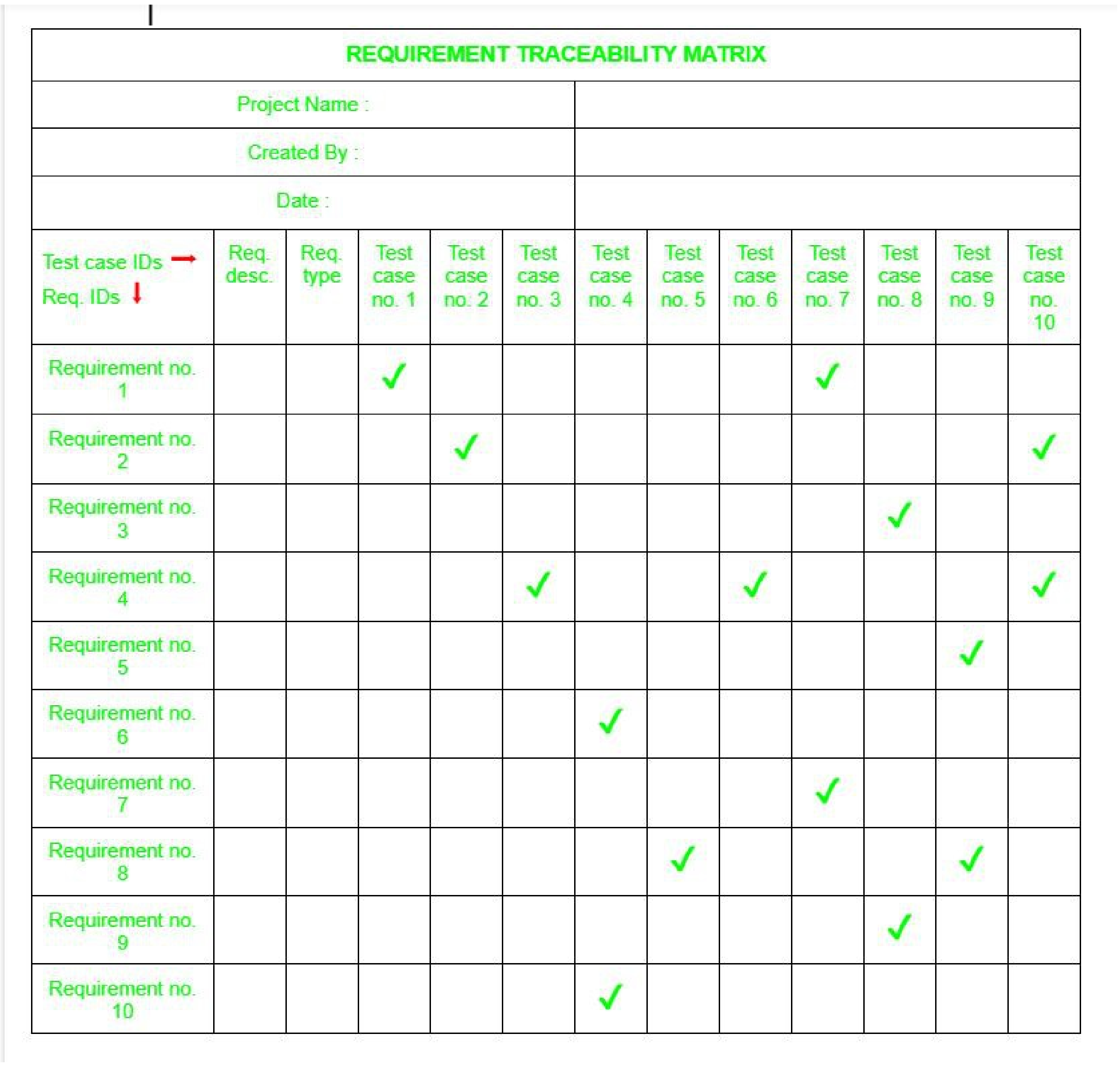
1. **Full test coverage:** RTM confirms the 100% test coverage.
2. **Verify missing functionality:** This document is helpful for the tester for checking there is not any functionality missed while testing the application.
3. **Helps to prioritize and track requirements:** It also helps to understand what extra test cases we added that are not part of the requirement.
4. **Helps to track test status:** It is easy to keep track of the overall test status.
5. **Proper consistent documentation:** RTM can help in the effort to provide proper and consistent documentation for the team.
6. **Versioning is easier:** RTM helps to keep track of the required modifications and how it impacts every part of the project.

RTM Template

The below figure shows the basic template of RTM. Here the requirement IDs are row-wise and test cases IDs are column-wise which means it is a forward traceability matrix.

From the figure below, it can be seen that:

- For verifying requirement number 1 there are test cases number 1 and 7.
- In requirement number 2 there are test cases number 2 and 10 and similarly, for all other requirements, there are test cases to verify them.



REQUIREMENT TRACEABILITY MATRIX												
Project Name :												
Created By :												
Date :												
Test case IDs → Req. IDs ↓	Req. desc.	Req. type	Test case no. 1	Test case no. 2	Test case no. 3	Test case no. 4	Test case no. 5	Test case no. 6	Test case no. 7	Test case no. 8	Test case no. 9	Test case no. 10
Requirement no. 1			✓						✓			
Requirement no. 2				✓								✓
Requirement no. 3										✓		
Requirement no. 4					✓			✓				✓
Requirement no. 5											✓	
Requirement no. 6						✓						
Requirement no. 7									✓			
Requirement no. 8							✓				✓	
Requirement no. 9										✓		
Requirement no. 10						✓						

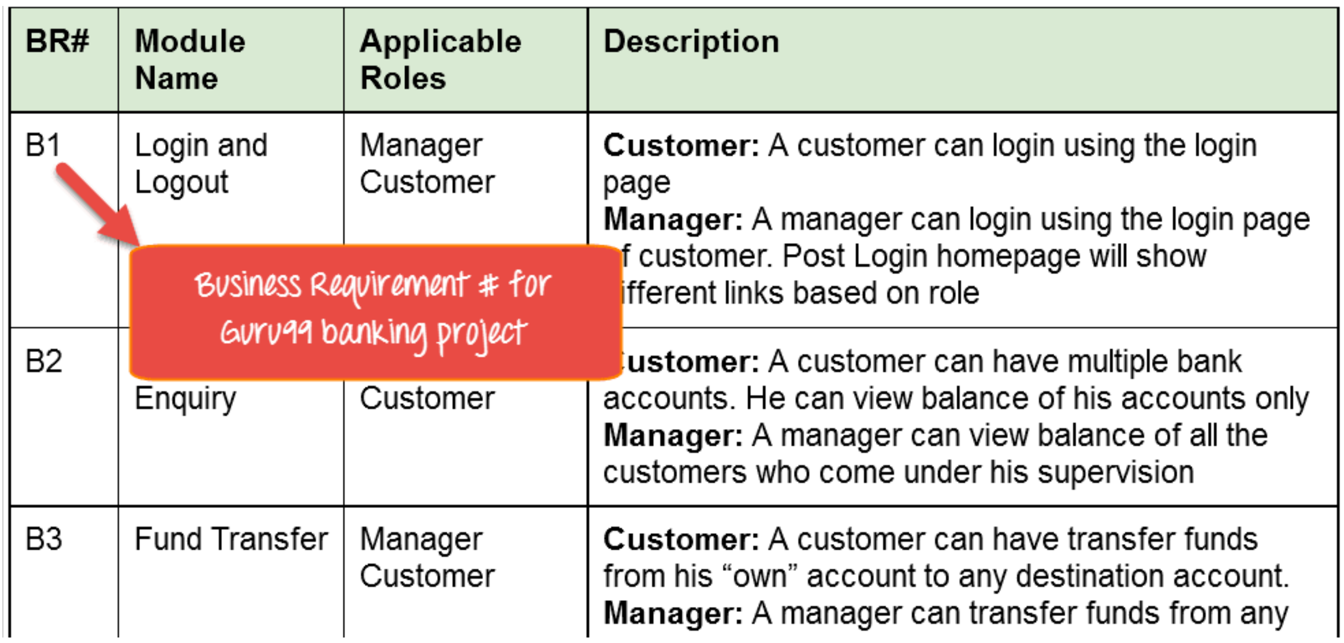
How to create Requirement Traceability Matrix

Let's understand the concept of Requirement Traceability Matrix through a Guru99 banking project.

On the basis of **the Business Requirement Document (BRD)** and **Technical Requirement Document (TRD)**, testers start writing test cases.

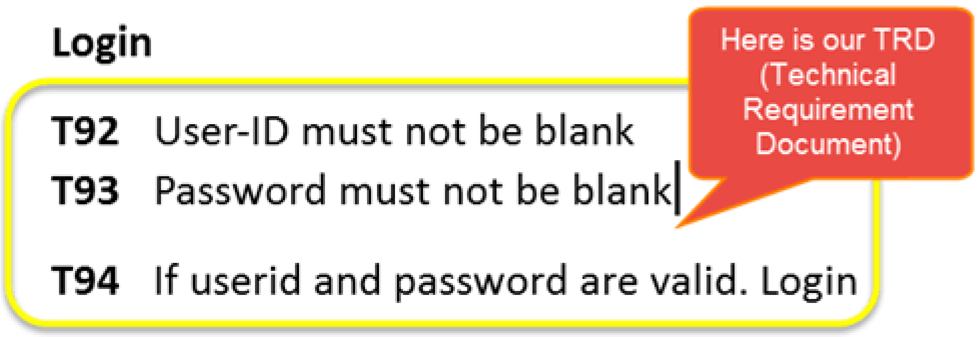
Let suppose, the following table is our Business Requirement Document or BRD for **Guru99 banking project**.

Here the scenario is that the customer should be able to login to Guru99 banking website with the correct password and user#id while manager should be able to login to the website through customer login page.



BR#	Module Name	Applicable Roles	Description
B1	Login and Logout	Manager Customer	**Customer:** A customer can login using the login page **Manager:** A manager can login using the login page f customer. Post Login homepage will show ifferent links based on role
B2	Enquiry	Customer	**ustomer:** A customer can have multiple bank accounts. He can view balance of his accounts only **Manager:** A manager can view balance of all the customers who come under his supervision
B3	Fund Transfer	Manager Customer	**Customer:** A customer can have transfer funds from his "own" account to any destination account. **Manager:** A manager can transfer funds from any

While the below table is our **Technical Requirement Document (TRD)**.



Login

T92	User-ID must not be blank
T93	Password must not be blank
T94	If userid and password are valid. Login

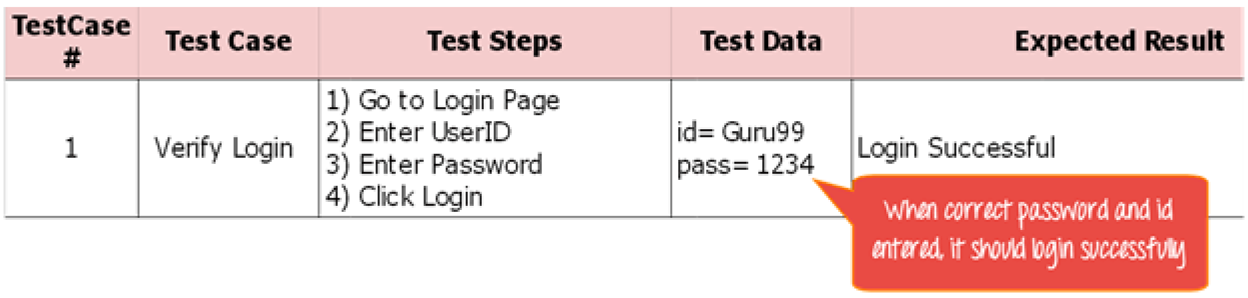
Here is our TRD (Technical Requirement Document)

Note: QA teams do not document the BRD and TRD. Also, some companies use **Function Requirement Documents (FRD)** which are similar to Technical Requirement Document but the process of creating Traceability Matrix remains the same.

Let's Go Ahead and create RTM in Testing

Step 1: Our sample Test Case is

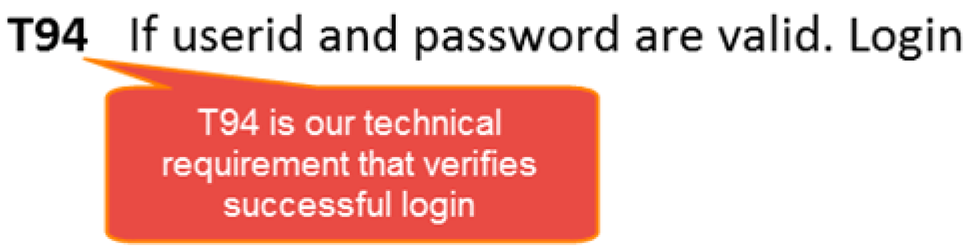
"Verify Login, when correct ID and Password is entered, it should log in successfully"



TestCase #	Test Case	Test Steps	Test Data	Expected Result
1	Verify Login	1) Go to Login Page 2) Enter UserID 3) Enter Password 4) Click Login	id= Guru99 pass= 1234	Login Successful

When correct password and id entered, it should login successfully

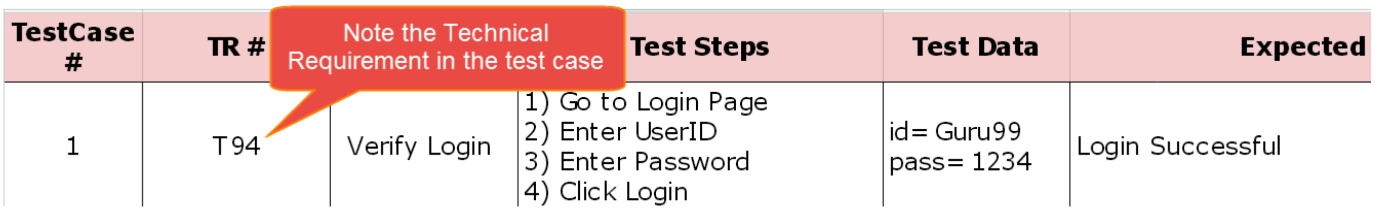
Step 2: Identify the Technical Requirement that this test case is verifying. For our test case, the technical requirement is T94 is being verified.



T94 If userid and password are valid. Login

T94 is our technical requirement that verifies successful login

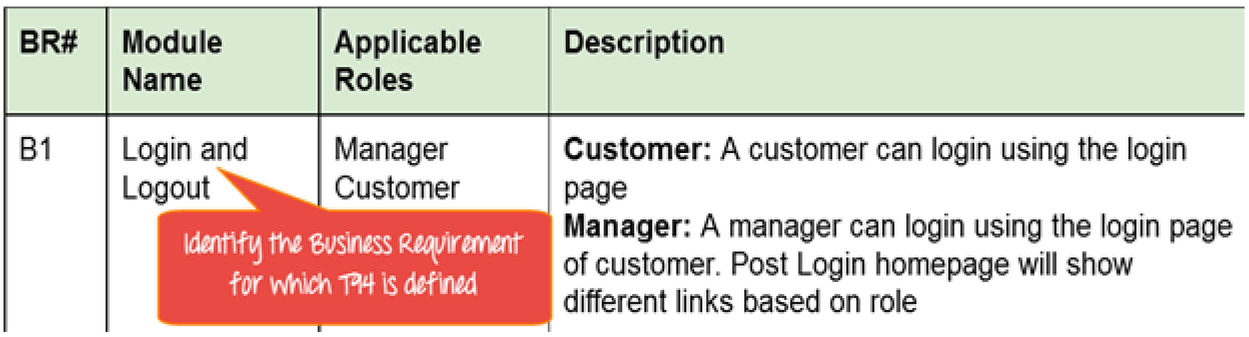
Step 3: Note this Technical Requirement (T94) in the Test Case.



TestCase #	TR #		Test Steps	Test Data	Expected
1	T94	Verify Login	1) Go to Login Page 2) Enter UserID 3) Enter Password 4) Click Login	id= Guru99 pass= 1234	Login Successful

Note the Technical Requirement in the test case

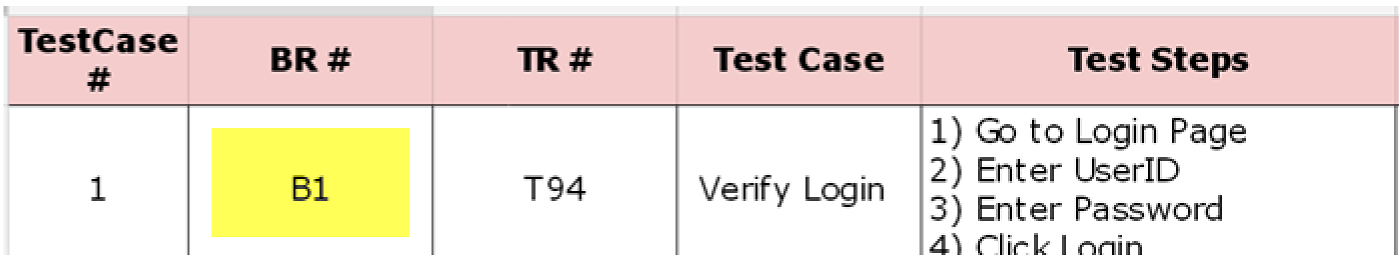
Step 4: Identify the Business Requirement for which this TR (Technical Requirement-T94) is defined



BR#	Module Name	Applicable Roles	Description
B1	Login and Logout	Manager Customer	**Customer:** A customer can login using the login page **Manager:** A manager can login using the login page of customer. Post Login homepage will show different links based on role

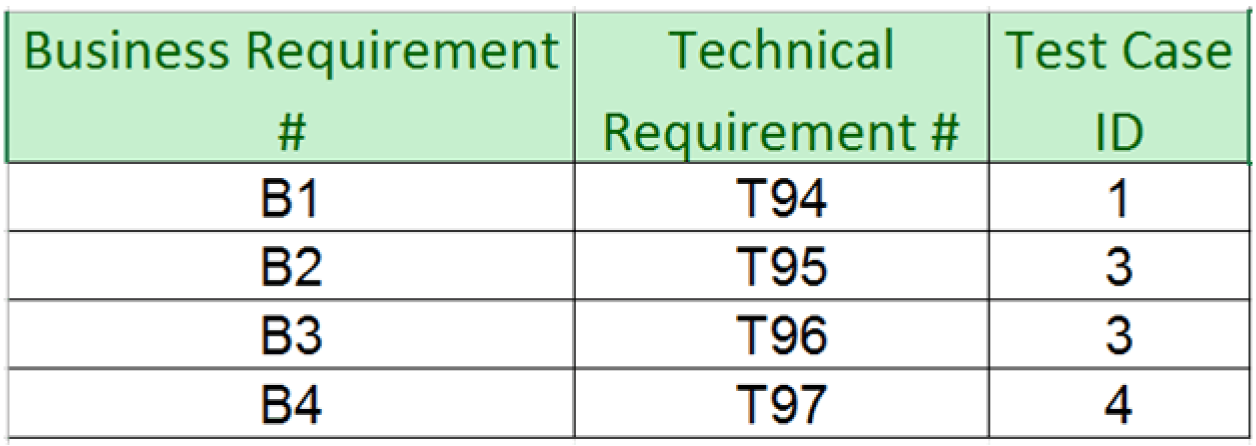
Identify the Business Requirement for which T94 is defined

Step 5: Note the BR (Business Requirement) in Test Case



TestCase #	BR #	TR #	Test Case	Test Steps
1	B1	T94	Verify Login	1) Go to Login Page 2) Enter UserID 3) Enter Password 4) Click Login

Step 6: Do above for all Test Cases. Later Extract the First 3 Columns from your Test Suite. RTM in testing is Ready!



Business Requirement #	Technical Requirement #	Test Case ID
B1	T94	1
B2	T95	3
B3	T96	3
B4	T97	4

Requirement Traceability Matrix

Advantage of Requirement Traceability Matrix

- It confirms 100% test coverage
- It highlights any requirements missing or document inconsistencies
- It shows the overall defects or execution status with a focus on business requirements
- It helps in analyzing or estimating the impact on the QA team's work with respect to revisiting or re-working on the test cases

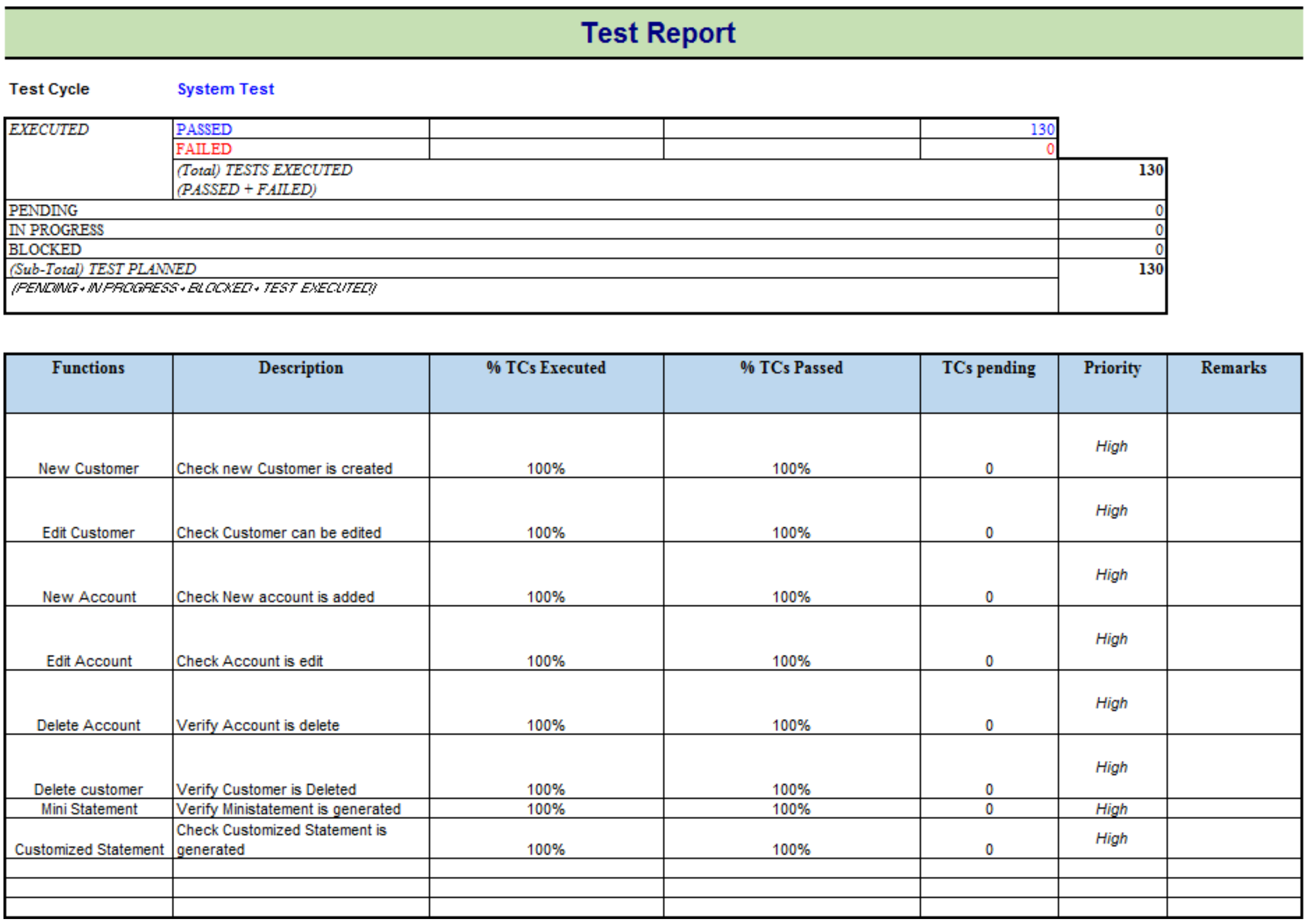
REQUIREMENTS TRACEABILITY MATRIX			
Project Name:			
Reviewer/Approver			
Traceability #	Requirement ID	Technical Requiremnt ID	Test Case ID
a	B1	T94	1
b	B1	T95	2
c	B3	T96	3
b	B4	T97	4

Test Report

Test Report is a document which contains a summary of all test activities and final test results of a testing project. Test report is an assessment of how well the Testing is performed. Based on the test report, stakeholders can evaluate the quality of the tested product and make a decision on the software release.

For example, if the test report informs that there are many defects remaining in the product, stakeholders can delay the release until all the defects are fixed.

Test Report Example



Test Report

Test Cycle System Test

EXECUTED	PASSED			130	
	FAILED			0	
	(Total) TESTS EXECUTED (PASSED + FAILED)				130
PENDING					0
IN PROGRESS					0
BLOCKED					0
(Sub-Total) TEST PLANNED (PENDING + IN PROGRESS + BLOCKED + TEST EXECUTED)					130

Functions	Description	% TCs Executed	% TCs Passed	TCs pending	Priority	Remarks
New Customer	Check new Customer is created	100%	100%	0	*High*	
Edit Customer	Check Customer can be edited	100%	100%	0	*High*	
New Account	Check New account is added	100%	100%	0	*High*	
Edit Account	Check Account is edit	100%	100%	0	*High*	
Delete Account	Verify Account is delete	100%	100%	0	*High*	
Delete customer	Verify Customer is Deleted	100%	100%	0	*High*	
Mini Statement	Verify Ministatement is generated	100%	100%	0	*High*	
Customized Statement	Check Customized Statement is generated	100%	100%	0	*High*	

Why Test Report?

The following scenario will show you why we do need the Test Report

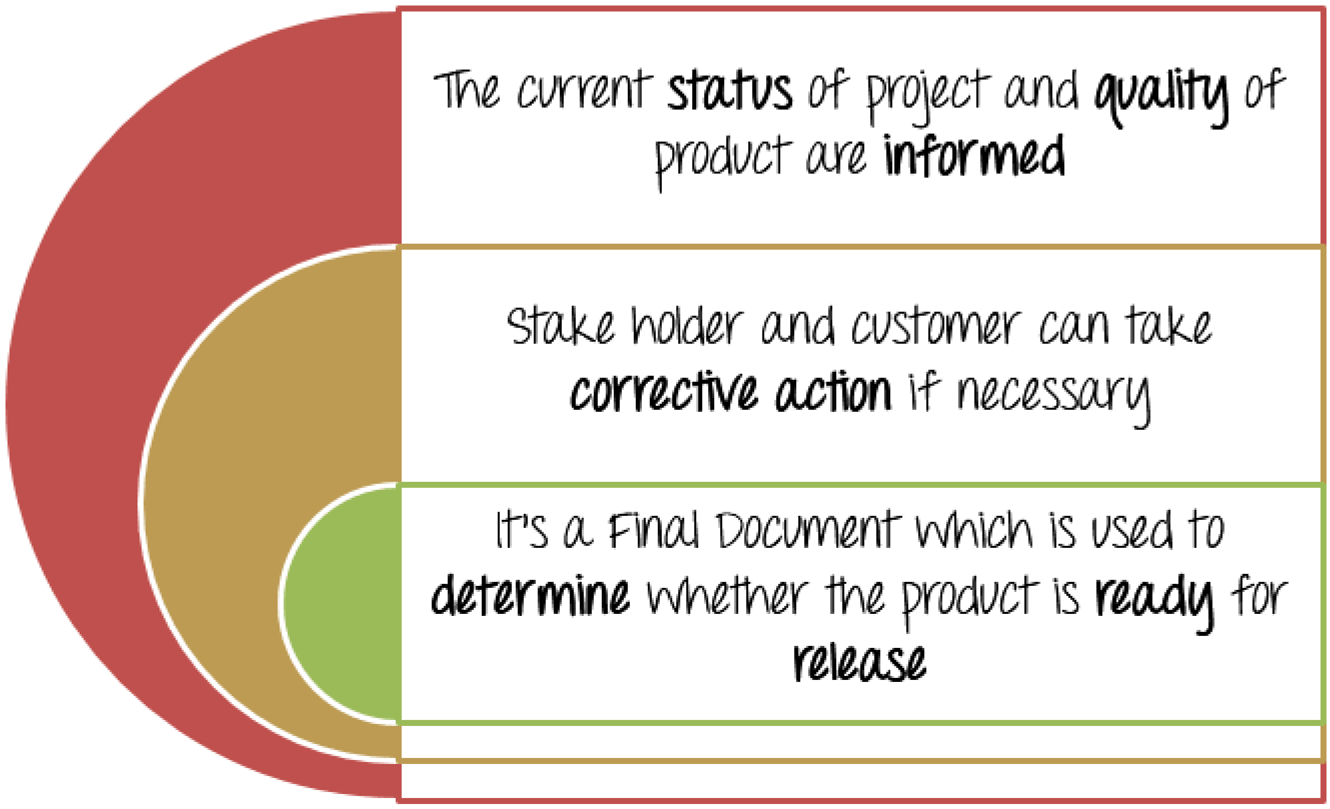
Earlier, when the boss asked you about whether the website Guru99 Bank can release, You answered him

The boss trusted you and decided to release this website to the customer at the end of the month. But 2 months post- release, you got the feedback from the client.

Do you know the root cause of this problem? Why does the website still has defects even when your Team has already tested it?

The problem is you ignored the reporting & evaluation phase in Test Management. The boss has no information to evaluate the quality of this website. They just trusted what you said and released the website without knowing its testing performance.

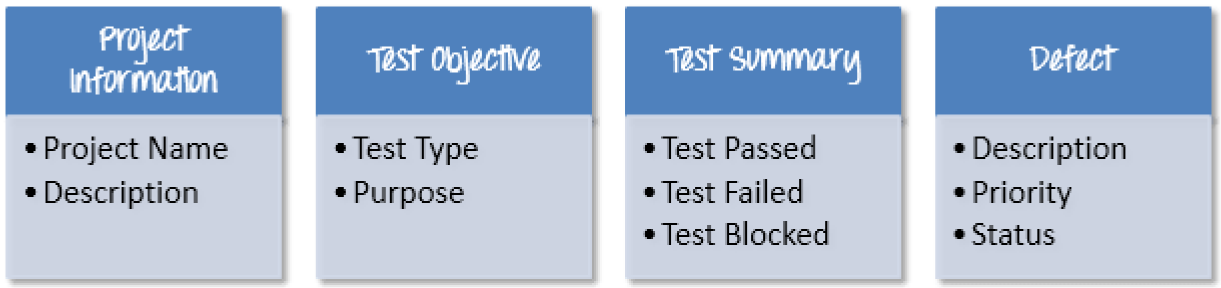
The typical benefits of a test report include:


The current status of project and quality of product are informed
Stake holder and customer can take corrective action if necessary
It's a Final Document which is used to determine whether the product is ready for release

How to make a good Test Report?

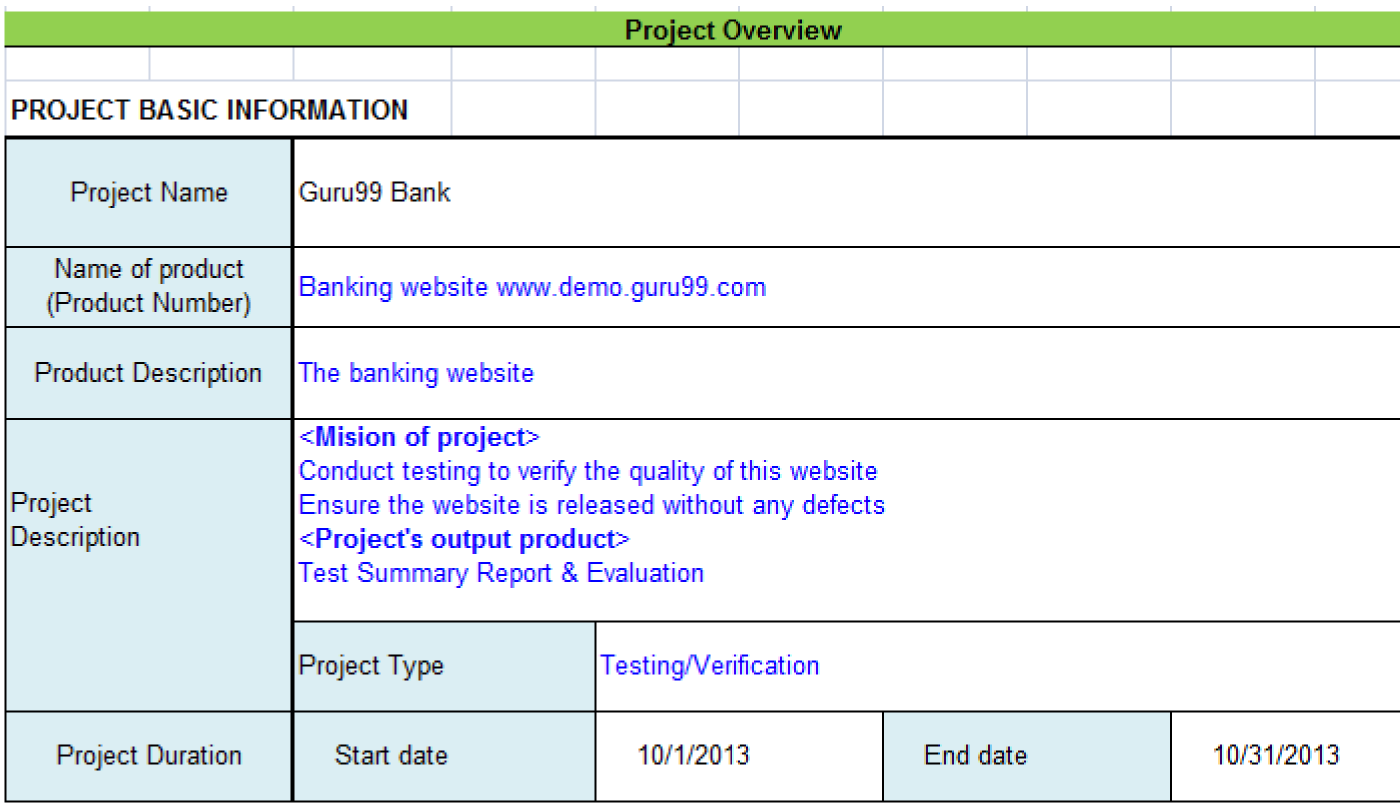
To answer this, you must know –

What does a test report contain?



Project Information

All information of the project such as the project name, product name, and version should be described in the test report. For example, the information of Guru99Bank project will be as follows



Project Overview

PROJECT BASIC INFORMATION

Project Name	Guru99 Bank			
Name of product (Product Number)	Banking website www.demo.guru99.com			
Product Description	The banking website			
Project Description	**<Mision of project>** Conduct testing to verify the quality of this website Ensure the website is released without any defects **<Project's output product>** Test Summary Report & Evaluation			
	Project Type	Testing/Verification		
Project Duration	Start date	10/1/2013	End date	10/31/2013

Test Objective

As mentioned in Test Planning tutorial, Test Report should include the objective of each round of testing, such as Unit Test, Performance Test, System Test …Etc.

Test Summary

This section includes the summary of testing activity in general. Information detailed here includes

- The number of test cases executed
- The numbers of test cases pass
- The numbers of test cases fail
- Pass percentage
- Fail percentage
- Comments

This information should be displayed **visually** by using **color indicator**, **graph, and highlighted table**.

Take a look at Test Report of the website Guru99 Bank to know more detail about Test report

Defect

One of the most important information in Test Report is defect. The report should contain following information

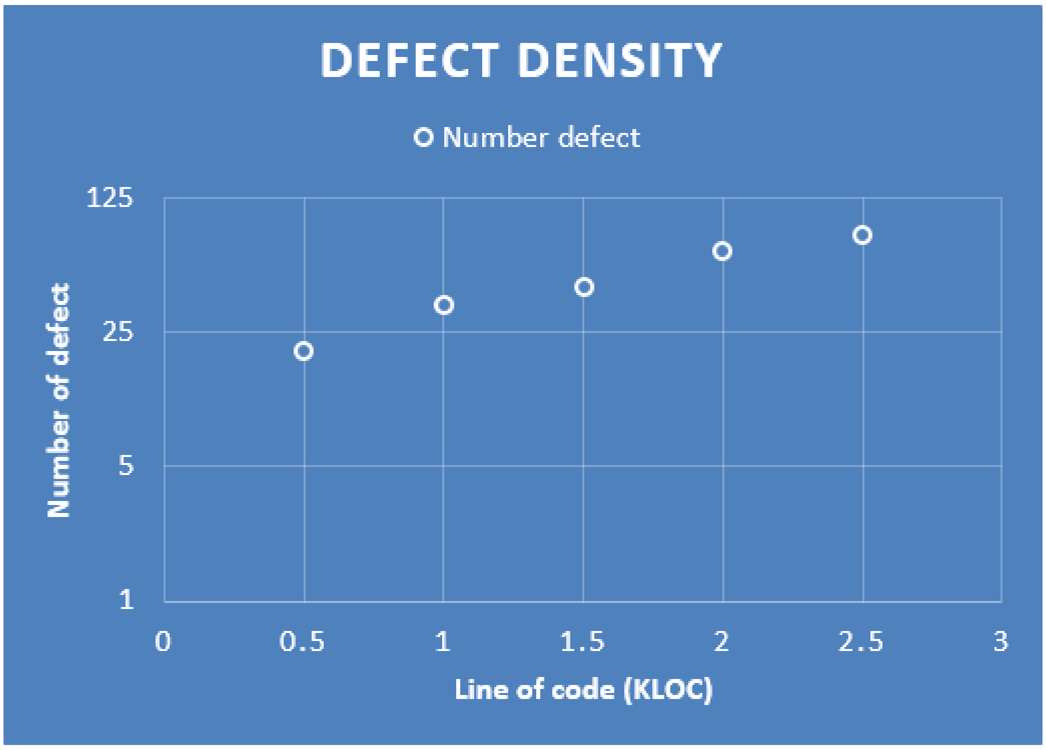
- Total number of bugs
- Status of bugs (open, closed, responding)
- Number of bugs open, resolved, closed
- Breakdown by severity and priority

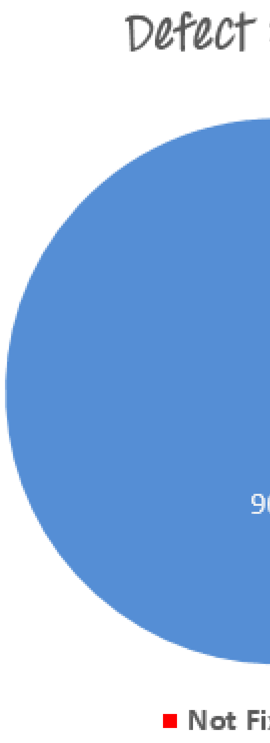
Like test summary, you can include some simple metrics like Defect density, % of fixed defects.

The project team sent you the Defect information as following

- Defect density is 20 defects/1000 lines of code average
- 90% defects fixed in total
- The detail of the bugs are described in this Defect tracker here

You can represent the data as following graph



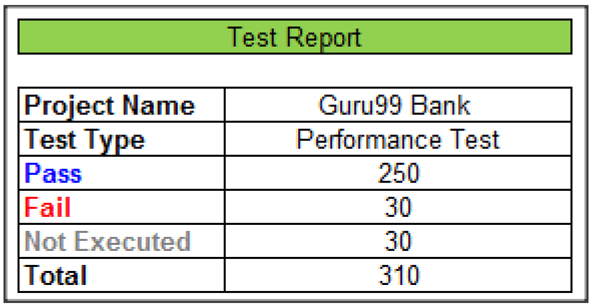


Tips to write a good test report

Test report is a **communication** tool between the Test Manager and the stakeholder. Through the test report, the stakeholder can **understand** the project situation, the quality of product and other things.

The following scenario shows you why we need a good Test Report

You co-operate with outsourcing company, its tester after having performed Performance Testing of the website Guru99 Bank, sends you a test report like this

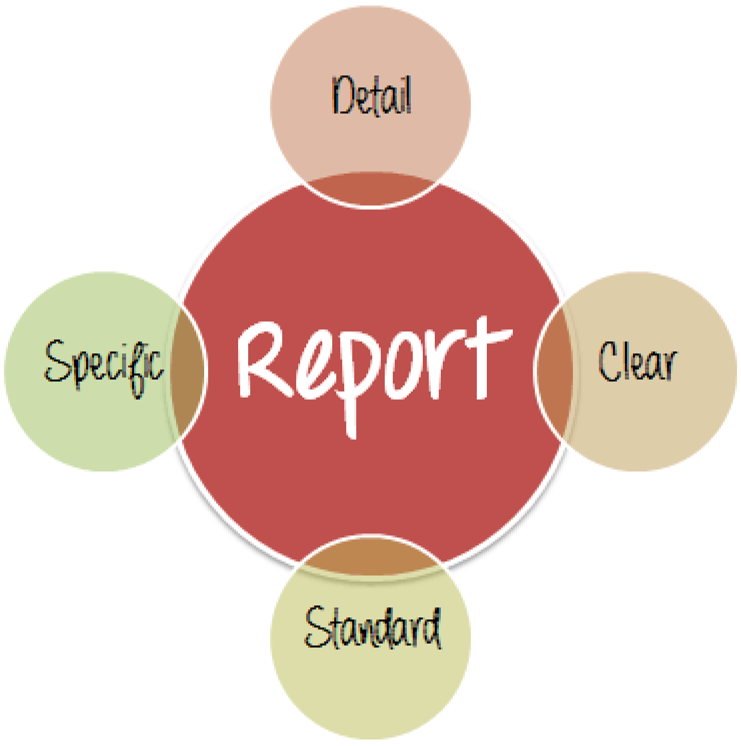


Test Report	
Project Name	Guru99 Bank
Test Type	Performance Test
Pass	250
Fail	30
Not Executed	30
Total	310

The information of that report is too **abstract**. It does not have any detailed information. The stakeholder who will read it might be slightly **puzzled** when they get it. They might ask or have following sets of questions: –

- Why did they not execute 30 TCs that remains
- What are these failed Test Cases
- Doesn't have any bugs description

To solve that problem, a good Test Report should be:



- **Detail**: You should provide a detailed description of the testing activity, show which testing you have performed. Do not put the abstract information into the report, because the reader will not understand what you said.
- **Clear:** All information in the test report should be **short** and **clearly** understandable.
- **Standard:** The Test Report should follow the **standard** template. It is easy for stakeholder to review and ensure the **consistency** between test reports in many projects.
- **Specific:** Do not write an essay about the project activity. Describe and summarize the test result specification and focus on the main point.

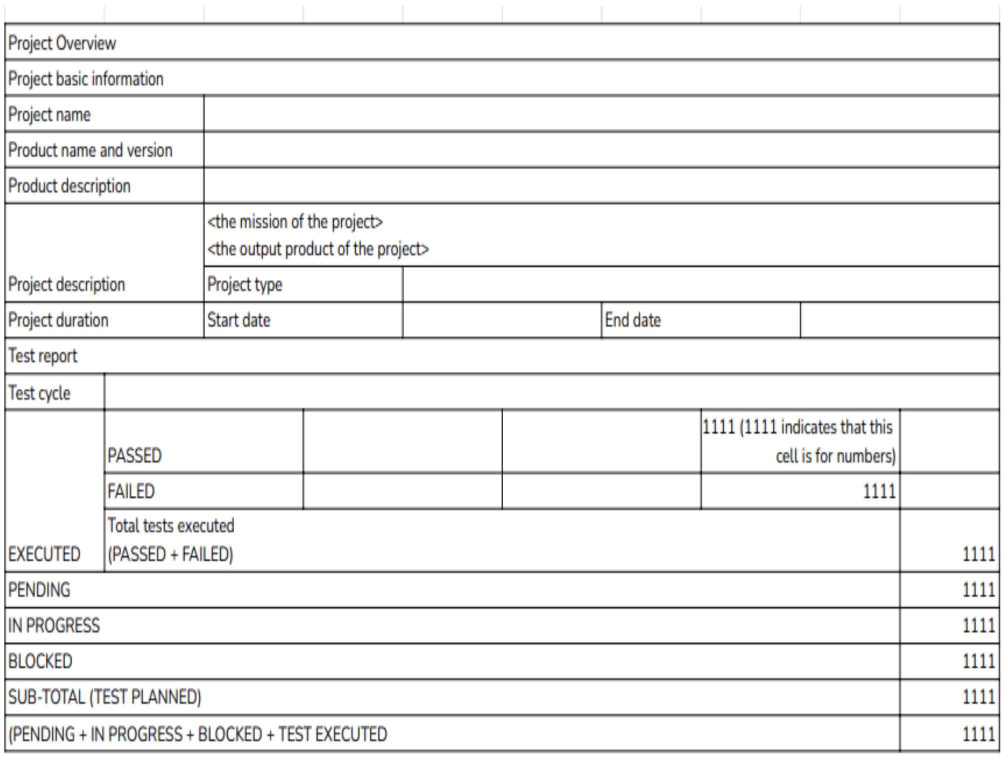
For example, to correct the above Test Report, the tester should provide more information such as:

- Project information
- Test cycle: (System Test, Integration Test…etc.)
- Which functions have already tested (% TCs executed, % TCs passed or fail…)
- Defect report (Defect description, Priority or status…)

Templates of a Test Report

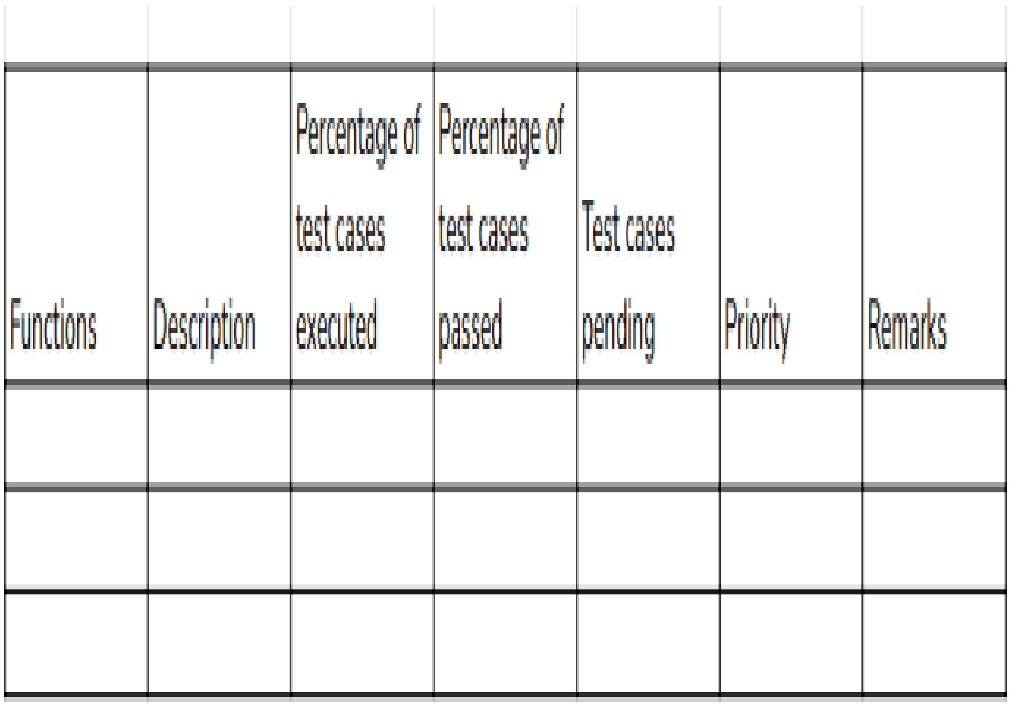
Here are some possible templates for a Test Report.

Template 1:



Project Overview					
Project basic information					
Project name					
Product name and version					
Product description					
Project description	<the mission of the project> <the output product of the project>				
	Project type				
Project duration	Start date		End date		
Test report					
Test cycle					
EXECUTED	PASSED			1111 (1111 indicates that this cell is for numbers)	
	FAILED			1111	
	Total tests executed (PASSED + FAILED)				1111
PENDING					1111
IN PROGRESS					1111
BLOCKED					1111
SUB-TOTAL (TEST PLANNED)					1111
(PENDING + IN PROGRESS + BLOCKED + TEST EXECUTED					1111

Template 2:



Functions	Description	Percentage of test cases executed	Percentage of test cases passed	Test cases pending	Priority	Remarks

References:

https://www.guru99.com/traceability-matrix.html

https://www.guru99.com/test-plan-v-s-test-strategy.html

https://www.guru99.com/how-test-reports-predict-the-success-of-your-testing-project.html