**Notes for MCA-II (Semester- III)**

**Subject :- Software Testing & Quality Assurance**

**(Subject Code:- IT-33)**

**Chapter 5 :- Test Management**

* **5.1. Test Organization- Roles & Skills of Tester, Test Lead, Test Manager :-**

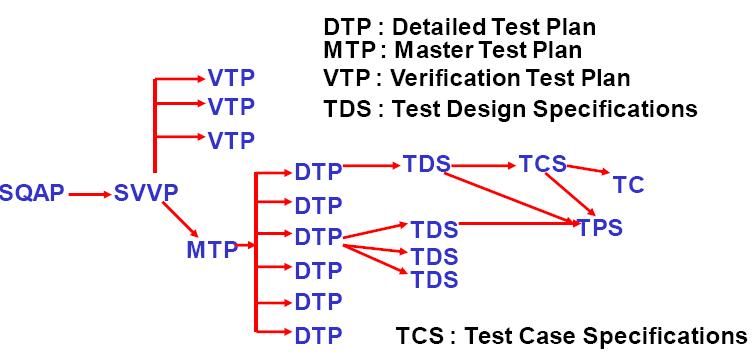
Software Testers are responsible for the quality of software development and deployment. They are involved in performing automated and manual tests to ensure the software created by developers is fit for purpose. Some of the duties include analysis of software, and systems, mitigate risk and prevent software issues.

**Test lead/Test Manager is responsible for:**

* Defining the testing activities for subordinates – testers or test engineers.
* All responsibilities of test planning.
* To check if the team has all the necessary resources to execute the testing activities.
* To check if testing is going hand in hand with the software development in all phases.
* Prepare the status report of testing activities.
* Required Interactions with customers.
* Updating project manager regularly about the progress of testing activities.
* Since the test manager represents the team he is responsible for all interdepartmental meetings.
* Interaction with the customers whenever required.
* A test manager is responsible for recruiting software testing staff. He has to supervise all testing activities carried out by the team and identify team members who require more training.
* Schedule testing activities, create budget for testing and prepare test effort estimations.
* Selection of right test tools after interacting with the vendors. Integration of testing and development activities.
* Carry out continuous test process improvement with the help of metrics.
* Check the quality of requirements, how well they are defined.
* Trace test procedures with the help of test traceability matrix.

**Test engineers/QA testers/QC testers are responsible for:**

* To read all the documents and understand what needs to be tested.
* Based on the information procured in the above step decide how it is to be tested.
* Inform the test lead about what all resources will be required for software testing.
* Develop test cases and prioritize testing activities.
* Execute all the test case and report defects, define severity and priority for each defect.
* Carry out regression testing every time when changes are made to the code to fix defects.
* analyzing users stories and/use cases/requirements for validity and feasibility
* collaborate closely with other team members and departments
* execute all levels of testing (System, Integration, and Regression)
* Design and develop automation scripts when needed
* Detect and track software defects and inconsistencies
* Provide timely solutions
* Apply quality engineering principals throughout the Agile product lifecycle
* Provide support and documentation
* A Software Tester is responsible for designing testing scenarios for usability testing.
* He is responsible for conducting the testing, thereafter analyze the results and then submit his observations to the development team.
* He may have to interact with the clients to better understand the product requirements or in case the design requires any kind of modifications.
* Software Testers are often responsible for creating test-product documentation and also has to participate in testing related walk through.
* **5.2 Test Plan:**
* A test plan outlines the strategy that will be used to test an application, the resources that will be used, and the test environment in which testing will be performed, the limitations of the testing and the schedule of testing activities.
* Typically the Quality Assurance Team Lead will be responsible for writing a Test Plan.
* General consideration of test planning is:
* Each testing activity has one or more inputs and consists of one or more tasks. Further each task produces one or more output.
* Planning is a document that should include the description of inputs, tasks to be carried out, output, scope of testing efforts, required resources, scheduling etc.



**Contents of high level test plan As Per BS –7925 & IEEE 829 standards:**

1. Test Plan Identifier 2.Introduction

3. Test Items 4.Features to be tested

5. Features not to be tested 6.Approach

7. Items Pass / Fail Criteria 8.Suspension / Resumption Criteria

9. Test Deliverables 10.Testing tasks

11. Environment 12.Responsibilities

13. Staffing & Training needs 14.Schedule

15. Risks & Contingencies 16.Approvals

**1. Test Plan Identifier:**

Unique reference number to this document.

**2. Introduction:**

A guide to tell that what the test plan covers, references to other relevant documents etc.

**3. Test Items:**

Physical things to be tested, such as executable programs, data files/databases. Their version no. with required details & references to relevant documentation should be there.

**4. Features to be tested:**

The logical things that are to be tested.

**5. Features not to be tested:**

Things (functions) not are to be tested

**6. Approach:**

The necessary activities to carry out the testing : tools, techniques, completion criteria ( such as coverage measures) & constraints such as environment restriction, staff availability etc.

**7. Items Pass / Fail Criteria:**

For each test item the criteria for passing or failing that item such as the number of known & Predicted outstanding faults.

**8. Suspension / Resumption Criteria:**

The criteria that will be used to determine when any testing activity should be suspended & resumed.

**9. Test Deliverables:**

What the testing process should provide in terms of documents reported i.e. test plans, test specifications, incident reports, summary reports etc.

**10. Testing tasks:**

Specific Tasks, Special Skills required & their interdependencies.

**11. Environment:**

Details of H/W& S/W + other facilities.

**12. Responsibilities:**

Who is responsible for what?

**13. Staffing & Training needs:**

Staff & their training on, to be tested system

**14. Schedule:**

Milestones w.r.t. the s/w delivery after testing, availability of test environment& test deliverables.

**15. Risks & Contingencies:**

What could go wrong & how can it be minimized.

**16. Approvals:**

**Case Study :- TEST PLAN FOR LIBRARY MANAGEMENT SYSTEM.**

The Library Management System (LMS) is an online application for assisting a librarian in managing a book library in a University. The system would provide basic set of features to add/update clients(Student, Staff), add/update books, search for books, and manage check-in / checkout processes. Our test group tested the system based on the requirement specification.

# Test Plan Identifier: LIBRARY MANAGEMENT SYSTEM

# INTRODUCTION:

This test report is the result for testing in the LMS. It mainly focuses on two problems: what we will test and how we will test.

|  |  |
| --- | --- |
| Prepared by: | Balaji Gawande |
| Version: | 1.1 |
| Revision: | 2 |
| Date: | July 09, 2021 |
| Project Manager: | Sachin Lende |
| Project Sponsor: | NTC |

# Test Items :

# System comprises mainly 3 modules: 1) Admin module, 2) Librarian/Student/Staff ,3)Inventory Items to be tested in these modules are:

* 1. Admin module: different command buttons like add project, add user, submit, cancel etc.
  2. Librarian/Student/staff User /login screen: login, pass word text boxes. Command buttons like submit, cancel, Forgot password etc,
  3. Inventory screen : where Liberian can ask Quotation, Purchase order, fill up A/c

details, Books/Journals/Papers Received Note.

* 1. **Features to be tested :**

1. Inside Admin module all command buttons, text boxes are to be tested.

2. Color, font size, length & other specified constraints about these boxes to be tested.

3. Form size, title validations & check other controls.

**5. Features not to be tested:**

All Non Functional Testing features.

# ITEM PASS/Fail CRITERIA

## GUI test

Pass criteria: librarians could use this GUI to interface with the backend library database without any difficulties

Result: pass

## Database test

Pass criteria: Results of all basic and advanced operations are normal

Result: pass

## Basic function test

### Add a student

Pass criteria:

* Each customer/student should have following attributes: Student ID (unique),Student Name, Class, Address and Phone number.

Result: pass

* The retrieved customer information by viewing customer detail should contain the four attributes.

Result: pass

### Update/delete student

Pass criteria:

* The record would be selected using the student ID

Result: pass

* Updates can be made on full. Items only: Class, Address, Phone number

Result: pass

* The record can be deleted if there are no books issued by user.

Result: Partially pass. When no books issued by user, he can be deleted. But when there are books Issued by this user, he was also deleted. It is wrong.(It’s Negative Test Case Example)

* The updated values would be reflected if the same customer's ID is called for.

Result: pass

* If customer were deleted, it would not appear in further search queries.

Result: pass

### Add a book

Pass criteria:

* Each book shall have following attributes: Book Number, ISBN, Title, Author name, Publication name with Edition

Result: pass

* The retrieved book information should contain the four attributes.

Result: pass

### Update/delete book

Pass criteria:

* The book item can be retrieved using the Book number

Result: did not pass. Can not retrive using the Book number

* The data items which can be updated is:- Edition No
* Result: pass
* The book can be deleted only if no user has issued it.

Result: partially pass. When no user has issued it, pass. When there are user having issued it,

did not pass

* The updated values would be reflected if the same call number is called for

Result: pass

* If book were deleted, it would not appear in further search queries.

Result: pass

### Search for book

Pass criteria:

* The product shall let Librarian query books’ detail information by their ISBN number or Author or Title.

Result: pass

* The search results would produce a list of books, which match the search parameters with following Details: Book number, ISBN number, Title, Author, Publication name with Edition

Result: pass

* The display would also provide the number of copies which is available for issue

Result: pass

* The display shall provide a means to select one or more rows to a user-list

Result: pass

* A detailed view of each book should provide information about check-in/check out status, with the borrower’s information.

Result: pass

* The search display will be restricted to 20 results per page and there would be means to navigate from sets of search results.

Result: pass

* The user can perform multiple searches before finally selecting a set of books for check in or checkout. These should be stored across searches.

Result: pass

* A book may have more than one copy. But every copy with the same ISBN number should have same detail information.

Result: pass

* The borrower’s list should agree with the data in students’ account

Result: pass

### Check-in book

Pass criteria:

* Librarians can check in a book using its call number

Result: pass

* The check-in can be initiated from a previous search operation where user has selected a set of books.

Result: pass

* The return date would automatically reflect the current system date.

Result: did not pass.

* Any late fees would be computed as difference between due date and return date at rate of 5 Rupees a day.

Result: did not pass

* A book, which has been checked in once, should not be checked in again

Result: pass

### Check-out book

Pass criteria:

* Librarians can check out a book using its Book number

Result: pass

* The checkout can be initiated from a previous search operation where user has selected a set of books.

Result: pass

* The student ID who is issuing the book would be entered

Result: pass

* The issue date would automatically reflect the current system date.

Result: did not pass

* The due date would automatically be stamped as 5 days from current date.

Result: did not pass

* A book, which has been checked out once, should not be checked out again

Result: pass

* A student who has books due should not be allowed to check out any books

Result: did not pass

* The max. No of books that can be issued to a customer would be 10. The system should not allow checkout of books beyond this limit.

Result: pass

### View book detail

Pass criteria:

* This view would display details about a selected book from search operation

Result: pass

* The details to be displayed are: Book number, IBN, Title, Author, Issue status (In library or checked out), If book is checked out it would display, User ID & Name, Checkout date, Due date

Result: for checkout date and due date, did not pass

* Books checked in should not display user summary

Result: pass

* Books checked out should display correct user details.

Result: pass

### View student detail

Pass criteria:

* Librarians can select a user record for detailed view

Result: pass

* The detail view should show:

a.User name, ID, Address & Phone number

Result: pass

b. The books issued by user with issue date, due date, call number, title

Result: did not pass

c. Late fees & Fines summary and total

Result: did not pass

* The display should match existing user profile

Result: pass

* The books checked out should have their statuses marked

Result: pass

* The book search query should show the user id correctly.

Result: pass

## Network test

Pass criteria: Results of operations (ping, ftp and ODBC connectivity check) are normal

Result: did not test this item, because no enough machines and no available environment.

# ENVIRONMENTAL

## Hardware: Pentium IV ,1 GB RAM,3.5 GZ Processor,etc …….

## Software: Microsoft Windows XP, Java, JDK, Apache,Tomcat, Oracle 10g,etc………..

## Browser :- IE 6.0 or Above,Google Chrome Ver (<< Version No >>),

## Mozilla Firefox(<< Version No >> ) ,etc………..

## 3.4 Testing Tool :- QTP 3.0 ,Load Runner 7.0,Test Director 5.0,Quality Center 6.0

# RESOURCES

## Developers of the system are involved in testing process (debugging, unit testing, Performance Testing, even integrity testing)

## Users of the system are involved in testing process (Customer Acceptance testing/User Acceptance testing)

## (Refer Point no **13 i.e. Responsibility**, to add more Resources )

# Scheduling, Staffing & Training :-

5.1 Java Development, Unit Testing & Debugging on Development server for LMS by first 2 week of march 2013 by Java Developer team leader and programmer having name :- <<name1,name 2,name3, etc…………>>

5.2 Integration testing, system testing, Performance Testing on Test Server & Automation Test Server for LMS by third week of march,2013 by QA Manager, Test Leader, Software Testers having name :- <<name1,name 2,name3, etc…………>>

5.3 User Acceptance Testing on UAT/CAT Server for LMS by first 4th week of march 2013 by End User, Client, Stake holder having name :- <<name1,name 2,name3, etc…………>>

5.4 Provide Advance Java ,Quick Test Professional (QTP) Training on Version <<Version no>>

On date 3/4/2013 to 6/4/2013 at Location <<XYZ Auditorium>> for LMS next Version <<Version no>> for Java Developer team leader and programmer , QA Manager, Test Leader, Software Testers

# Approach

### **Test planning activity**

In preparation for testing, the following activities will occur:

* Development and approval of the Online Library System <<Version No>> of test plan.
* Preparation and organization of test data within TestDirector ( Automation tool).
* Identification and review of defect corrections to be included in the release.
* TestDirector ( Automation tool).project environment created with appropriate users.
* Unit testing
* Integration testing
* Validation testing
* Regression Testing
* Security & System Testing
* Acceptance Testing

### **Perform testing activity**

**Databases**

Each test executor will be responsible for:

* Running their own batches.
* All system test activity will be conducted on “Online Library << Version No>> ” database.
* Creating their own test data.

# 

# Test Deliverables

## *Test Results*

The following test result deliverables will be produced:

* Daily Status Reports
* Test summary report
* Updates to the defect tracking system (defects)

# Testing tasks

Project Manager, QA Manager will have to Communicate Subject Matter Expert and Have to Learn Library Domain. Sr. Tester have to learn QTP, Test Director and other tools need for Customize online Library management System .Testing for Version 1.1 should be completed before 1 week to the version 1.2 Design Document .

# Responsibilities

1. Project Manager along with Onsite Coordinator, Sr. Designer and Domain Expert will take the Requirement Analysis, feasibility study to Prepare Design Document .
2. QA Manager will Create Test Plan along with Designer within a week after Requirement Analysis
3. 2- Tester will work to prepare Test cases on basis of Test Plan on Test Server .
4. 3- Programmer will prepare Unit test case along with Code on Development Server .
5. 1-Tester will work for Automation Testing tool i.e. Test Director 5.0 on Automation Test Server.
6. 1- Tester will work on Customer Acceptance Test Server & Integration Test Server .
7. 1-Configuration Manager will manage all Version Control And Change control time to time from all Test Server & Development Server.
8. 1-Network Manager will Manage all Network Configuration, Network Conjuction
9. 1-DBA will manage project database and create memory spaces for every iterations and versions of project.

# Suspension & Resumption Criteria

1. Tester’s have to upload all Test cases and Test Review only on Test Server They can suspend all work at the end of day by logging off from server and resumption by adding his own Employee id and password from secure id and have to run Script on server end for Login.
2. Programmer have to upload all Code , Unit Test cases and Code Review only on Development Server They can suspend all work at the end of day by logging off from server and resumption by adding his own Employee id and password from secure id and have to run Script on server end for Login

# Risks & Contingencies

1. Converting User Requirement into Technical Requirement.
2. Loss of Financial & Sensitive Data.
3. Lack of Inappropriate Back-up & Recovery Plan.

**Please see below table other types of risks & contingencies plan**

|  |  |  |  |
| --- | --- | --- | --- |
| **Priority** | **Specific Risk Item** | **Probability of Occurrence**  **(%)** | **Contingency** |
| 1 | Greater than 15% error rate occurs during QA Test due to lack of non-author testing | 60% | Increase the number of testers on the project. |
| 2 | Test cases do not provide clear and accurate test steps for the test team. | 50% | Contact test team for further clarification of test cases. |
| 3 | Environmental downtime/delays. | 15% | Add more DBA support. |

# Approvals

|  |  |  |
| --- | --- | --- |
| **Designation** | **Approval Task** | **Sign-off along with date** |
| Delivery Head | will get Approval from Management & Client for Creating Secure Id password on Client Server |  |
| Project Manager | a)Will Get Approval from Trainer for their subordinate’s Training Schedule  b) Will get Delivery Head Approval for subordinate’s any leave  c) Will get Approval from DBA Head for Memory Space and other DB activity for Project. |  |
| QA Manager | Will Get Approval from Test Tool Vendor for QTP, Test Director Testing Tool |  |

* **5.3. Test Process Monitoring & Control**
* **5.3.1. Test Monitoring through -Test Log and Defect Density**

Test log is one of the crucial test artifact prepared during the process of testing. It provides a detailed summary of the overall test run and indicates the passed and failed tests. Additionally, test log also contains details and information about various test operations, including the source of issues and the reasons for failed operations. The focus of this report/document is to enable post execution diagnosis of [failures and defects](https://www.professionalqa.com/defect-and-failure) in the software.

## **Test Log Template:**

* **Test Log Identifier**: This is a distinctive company allocated number, that offers assistance in identifying the test log, its level, and the level of the software it is related to. This number usually acts as a short name of the test log. Apart from the number, it also includes the version date and version number of the log.
* **Description**: Once a unique identifier is allocated to the software, the team offers a detailed description of the various items being testing and any supporting reference material. Other details included here are:
  + Date and time.
  + Executed by.
  + Environment.
  + Case specifications.
  + Procedural specifications.
* **Activity & Event Entries**: In this section of the report, the team defines the various activities performed by them as well as the multifarious event entries, which are used for test execution. This portion is further divided into five parts, which are:
  + **Execution Description**: Offers information about the procedure executed, the individuals present during the testing activities, and the reasons for their presence during the process of testing.
  + **Procedure Results**: Here, all the relevant information is logged for each execution along with results status.
  + **Environmental Information**: In this section, any changes or substitutions from the requested environment is provided by the team.
  + **Anomalous Events**: The events are recorded before and after the occurrence of anomalies (Difference or Variances) and errors.
  + **Incident Report**: At the end of the test log, any variances and difference in the output are reported.

## **Defect Density :-**

Software is tested based on its quality, scalability, features, security, and performance, including other essential elements. It's common to detect defects and errors in a software testing process. However, developers must ensure they are taken care of before launching it to the end-users. This is because fixing an error at an early stage will cost significantly less than rectifying it at a later stage.

The process of defect detection ensures developers that the end product comprises all the standards and demands of the client. To ensure the perfection of software, software engineers follow the defect density formula to determine the quality of the software.

**More Defects = Lower Quality**

## What is Defect Density in software testing

Defect density is numerical data that determines the number of defects detected in software or component during a specific development period. It is then divided by the size of the software. In short, it is used to ensure whether the software is released or not.

The role of defect density is extremely important in Software Development Life Cycle (SDLC). First, it is used to identify the number of defects in software. Second, this gives the testing team to recruit an additional inspection team for re-engineering and replacements.

Defect density also makes it easier for developers to identify components prone to defects in the future. As a result, it allows testers to focus on the right areas and give the best investment return at limited resources.

## How to Calculate Defect Density?

The defect density is calculated by dividing the 'total defects' of software by its 'Size.'

**Defect Density = Total Defect/Size**

According to best practices, one defect per 1000 lines (LOC) is considered good. Such standard of defect density is called KLOC. The size of the software or code is expressed in Function Points (FP).

## Steps to calculate Defect Density −

Collect the total defects detected during the software development process

**Calculate Defect Density = Average number of Defects/KLOC**

Let's understand it with an example −

Let's say your software comes with five integrated modules.

* Module 1 = 5 bugs
* Module 2= 10 bugs
* Module 3= 20 bugs
* Module 4= 15 bugs
* Module 5= 5 bugs
* Total bugs = 5+10+20+15+5= 55

Now total line of code for each module is

* Module 1= 500 LOC
* Module 2= 1000 LOC
* Module 3= 1500 LOC
* Module 4= 1500 LOC
* Module 5= 1000 LOC

Total Line of Code = 500 + 1000 + 1500 + 1500 + 1000 = 5500

Defect Density = 55/5500 = 0.01 defects/LOC or 10 defects/KLOC

## Uses of Defect Density

Defect density is considered an industry standard for software and its component development. It comprises a development process to calculate the number of defects allowing developers to determine the weak areas that require robust testing.

Organizations also prefer defect density to release a product subsequently and compare them in terms of performance, security, quality, scalability, etc. Once defects are tracked, developers start to make changes to reduce those defects. The defect density process helps developers to determine how a reduction affects the software quality-wise.

The use of defect density is inconsiderable in many ways. However, once developers set up common defects, they can use this model to predict the remaining defects. Using this method, developers can establish a database of common defect densities to determine the productivity and quality of the product.

## Factors Affecting Defect Density Metrics

As we know, defect density is measured by dividing total defects by the size of the software. The goal is not about detecting the defects but to detect defects that actually matter. Therefore, it's crucial to understand the factors that result in an efficient outcome. Developers and the testing team need to arrange all the necessary conditions before initiating this process. This helps developers trace the affected areas properly, allowing them to achieve highly accurate results.

Factors that affect defect density are −

* Types of defects
* Critical and complexity of the code used
* Skills of the developer and testing team
* Time allocated to calculate the defect density

Above all, the efficiency and performance of the software remain the biggest factor that affects the defect density process.

## Advantages of Defect Density

Defect density comes with several benefits for software testers and developers. Apart from providing exceptional accuracy in defect measurements, it also caters to many technical and analytical requirements. Having accurate results at hand can help software engineers stay confident about their developed software's quality and performance.

## Several other advantages of defect density include −

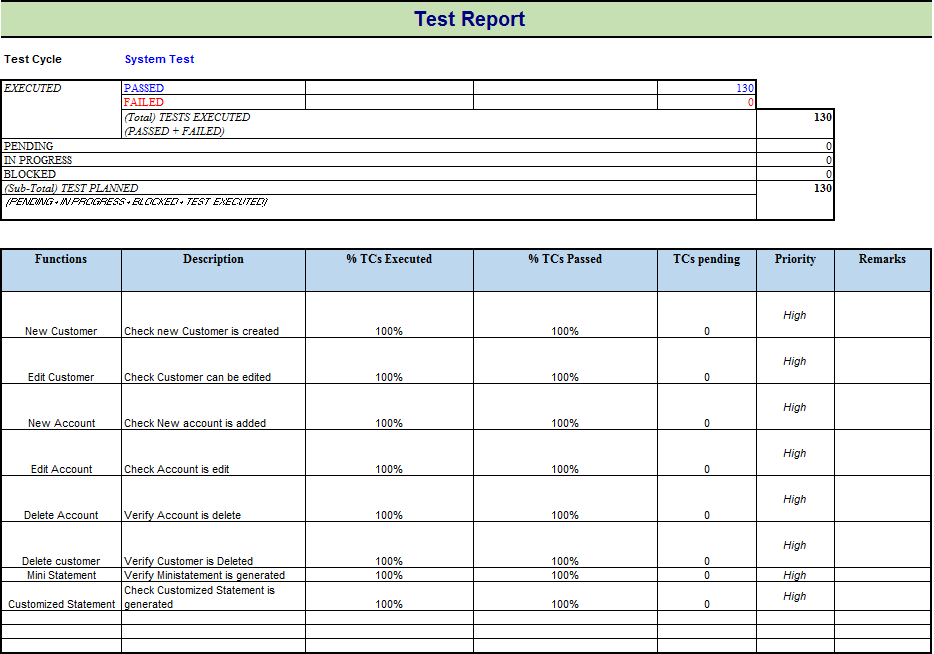
* Developers can ensure that the product set to launch doesn't require any more testing
* Developers and testers can estimate the testing and rework required to fix the errors
* Testers can trace and detect components possessing high risks
* The testing team can determine the amount of training requires to complete the testing process
* One can identify the area of improvement and fix it
* **5.3.2. Reporting Test Status :-**

**(TEST SUMMARY REPORT TEMPLATE):-**

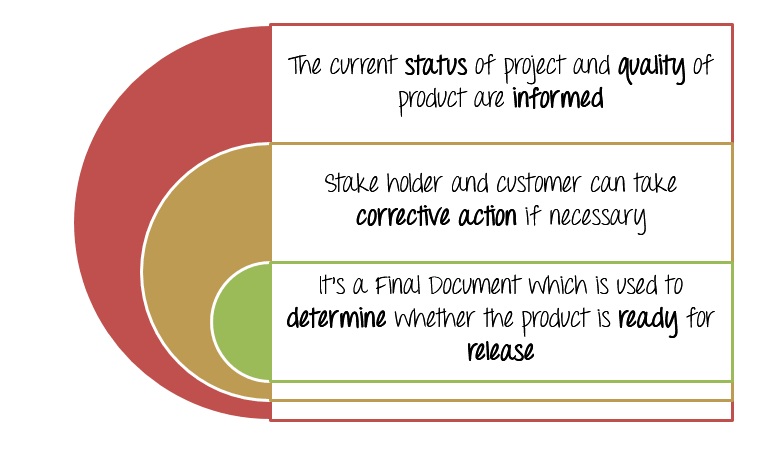
Test Report is a document which contains a summary of all test activities and final test results of a testing project. Test report is an assessment of how well the[Testing](https://www.guru99.com/software-testing.html)is performed. Based on the test report, stakeholders can evaluate the quality of the 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.

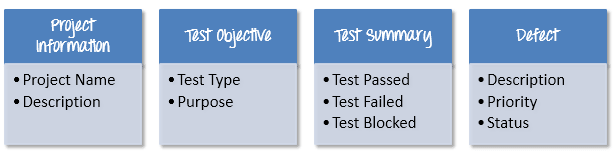
**Test Report Example**



**The typical benefits of a test report include:**

****

## **What does a test report contain?**



### **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

### **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
* **5.3.3 Test Control :-**

## **What is Test Monitoring?**

**Test Monitoring** in test execution is a process in which the testing activities and testing efforts are evaluated in order to track current progress of testing activity, finding and tracking test metrics, estimating the future actions based on the test metrics and providing feedback to the concerned team as well as stakeholders about current testing process.

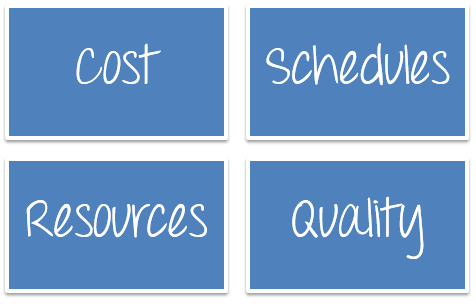
## **What is Test Control?**

**Test Control** in test execution is a process of taking actions based on results of the test monitoring process. In the test control phase, test activities are prioritized, test schedule is revised, test environment is reorganized and other changes related to testing activities are made in order to improve the quality and efficiency of future testing process.

## **What do we monitor?**

Monitoring will allow you to make comparisons between your original plan and your progress so far. You will be able to implement changes, where necessary, to complete the project successfully.

In your project, as the Test Manager, you should monitor the key parameters as below



**Costs** :- are an important aspect of project monitoring and control. You have to estimate and track basic cost information for your project. Having accurate project estimates and a robust project budget is necessary to deliver project within the decided budget.

Suppose, your boss has agreed to fund the project with $100,000. You must keep an eye on the actual costs while the project is being implemented.

**The schedule tells you:-**

When should each activity be done?

What has already been completed?

The sequence in which things need to be finished.

## **Resources:-**

As mentioned in previous articles, **resources** are all things required to carry out the project tasks. They can be people or equipment required to complete the project activity. Lack of resources can affect the project progress.

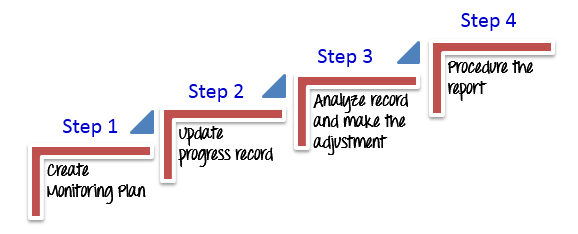
The truth is, everything may not happen as planned, employees will leave, the project budget may be cut, or the schedule will get pushed. Monitoring resources will help you to early detect any resource crunch and find a solution to deal with it.

## **Quality:-**

Quality monitoring involves monitoring the results of specific **work products** (like test case suite, test execution log), to evaluate whether its meets the defined quality standards. In case results do not meet quality standards, you need to identify potential resolution.

## How to monitor?

To monitor project progress effectively, you should follow the following steps



**Step 1) Create Monitoring Plan:-**

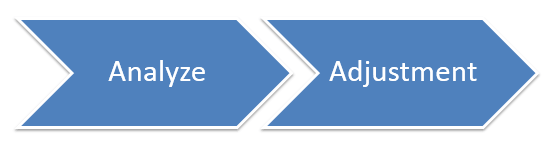
* When to collect data ?
* What data need to collect and measure?
* How to evaluate Project Progress via metrics ?
* What metrics you need to collect and measure?
* When to collect the metrics?
* How to evaluate the project’s progress via metrics?

## **Step 2) Update progress record**

* With time, your team member will be making progress on their project task. You must track their activity as per schedule and ask them frequently update the progress information such as time spent, task status…etc. By checking these records, you can immediately see the impact on the project plan.

## **Step 3) Analyze record and make the adjustment**

## **There’re 2 sub-steps in the steps**



## **Step 3.1)**

## **Analyze**

In this step, you compare the progress you defined in plan with the actual progress that the team has made. By analyzing the record, you can also see how much time has been spent on individual task and the total time spent on the project overall.

By tracking and analyzing the project progress, you can early detect any issue which may happen to the project, and you can find out the solution to solve that issue.

## **Step 3.2) Adjustment**

Make the necessary adjustments keep your project on track. Reassign tasks, modify schedules, or reassess your goals. This will help you keep moving toward the finish line.

## **Step 4) Produce the report ( A Template )**

|  |
| --- |
| Company Name |
| Infosys ©  PROJECT REPORTING |
| UBS Bank |

* 1. VERSION HISTORY

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Version #** | **Write by** | **Revision**  **Date** | **Approved**  **By** | **Approval**  **Date** | **Outline** |
| 1.0 | *<Author name>* | *<mm/dd/yy>* | *<name>* | *<mm/dd/yy>* | Project Report create |
|  | Sachin Lende | 02/23/2022 | Sanjay Solanki | 02/28/2022 |  |
|  |  |  |  |  |  |

* 1. PROJECT OUTLINE

|  |  |
| --- | --- |
| Project Name | UBS Bank |
| Project Description | Test the net banking facility of the esteemed “UBS Bank” |
| Project Director | Sanjay Solanki |
| Date approved | 02/28/2022 |
| Estimated completion date | 03/28/2022 |

* 1. CURRENT COST

|  |  |
| --- | --- |
| Estimated total cost | 100,000$ |
| Initial cost estimate | 40,000$ |
| Total spend to date | 42,000$ |
| Planned spend to date | 50,000$ |
| Forecast spend | 100,000$ |

* 1. **Current Schedule**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Task Name** | **Start Date** | **End Date** | **Duration** | **% Complete** | **Date** | **Date** | **Date** | **Date** | **Date** |
| Test Case Writing |  |  |  |  |  |  |  |  |  |
| Test Case Review |  |  |  |  |  |  |  |  |  |
| Test Case Execution |  |  |  |  |  |  |  |  |  |

* 1. **Current Resources**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No** | **Name** | **Quantity** | **Status** | **Remark** |
| 1 | Web Server | 1 | Good |  |
| 2 | Database server | 1 | Good |  |
| 3 | Tester | 3 | Lack of Human Resource | 1 tester is on 3 days leave |
| 4 | Developer | 4 | Ok |  |

* 1. **Current Quality :-**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test Case** | **Pass** | **Failed** | **Total Executed** | **Total Test Cases** | **Pass Rate** | **Run Rate** |
| Pass | 100 | 10 | 110 | 130 | 90% | 84% |
|  |  |  |  |  |  |  |

**Defect Metrics**

|  |  |
| --- | --- |
| Defect Rejection Ratio | 23.8 % |
| Defect Leak Ratio | 31.2 % |
| Total Defect | 64 |

1. **Current Critical Issue & Decision Required**

*NIL*

**5.4 Requirement Traceability Matrix (RTM) :-**

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.

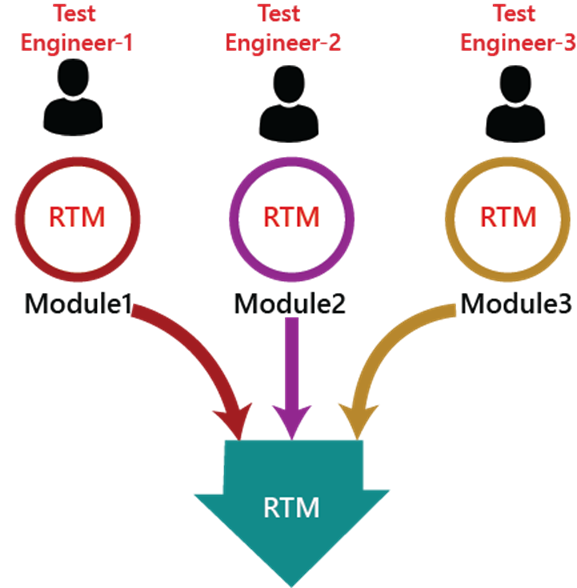
Traceability matrix is a table type document that is used in the development of software application to trace requirements. It can be used for both forward (from Requirements to Design or Coding) and backward (from Coding to Requirements) tracing. It is also known as **Requirement Traceability Matrix (RTM) or Cross Reference Matrix (CRM).**

**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.

It is prepared before the test execution process to make sure that every requirement is covered in the form of a Test case so that we don't miss out any testing. In the RTM document, we map all the requirements and corresponding test cases to ensure that we have written all the test cases for each condition.

**The test engineer** will prepare RTM for their respective assign modules, and then it will be sent to the Test Lead. The Test Lead will go repository to check whether the Test Case is there or not and finally Test Lead consolidate and prepare one necessary RTM document.

This document is designed to make sure that each requirement has a test case, and the test case is written based on business needs, which are given by the client. It will be performed with the help of the test cases if any requirement is missing, which means that the test case is not written for a particular need, and that specific requirement is not tested because it may have some bugs. The traceability is written to make sure that the entire requirement is covered.



We can observe in the below image that the requirement number 2 and 4 test case names are not mentioned that's why we highlighted them, so that we can easily understand that we have to write the test case for them.

Generally, this is like a worksheet document, which contains a table, but there are also many user-defined templates for the traceability matrix. Each requirement in the traceability matrix is connected with its respective test case so that tests can be carried out sequentially according to specific requirements.

## **Why RTM is Important?**

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.’

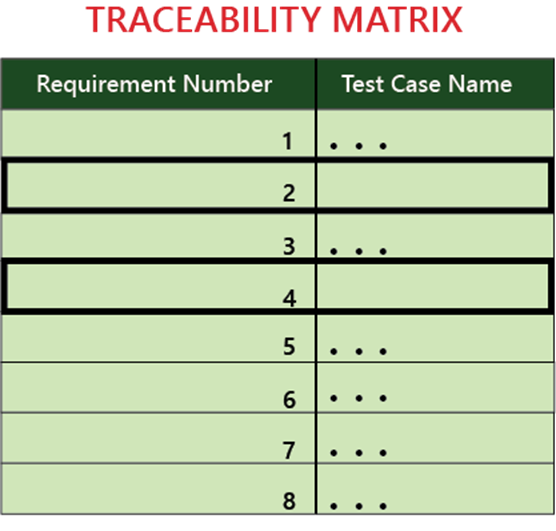
The traceability matrix is typically a worksheet that contains the requirements with its all possible test scenarios and cases and their current state, i.e. if they have been passed or failed. This would help the testing team to understand the level of testing activities done for the specific product.

**Note:**

We go for RTM after approval and before execution so that we don't miss out on any Test Case for any requirement.

We don't write RTM while writing the testing because it can be incomplete, and after writing the test case, we don't go here because the test case can be rejected.

RTM document ensures that at least there is one test case written in each requirement, whereas it does not talk about all possible test cases written for the particular requirement.

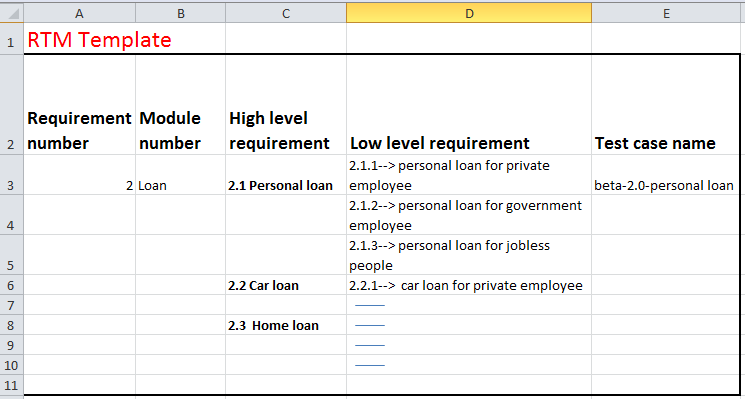


## **RTM Template**

Below is the sample template of requirement traceability matrix (RTM):

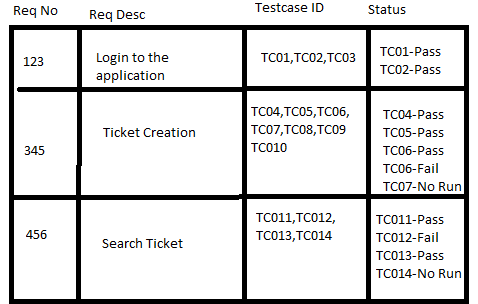


**RTM Example :-**



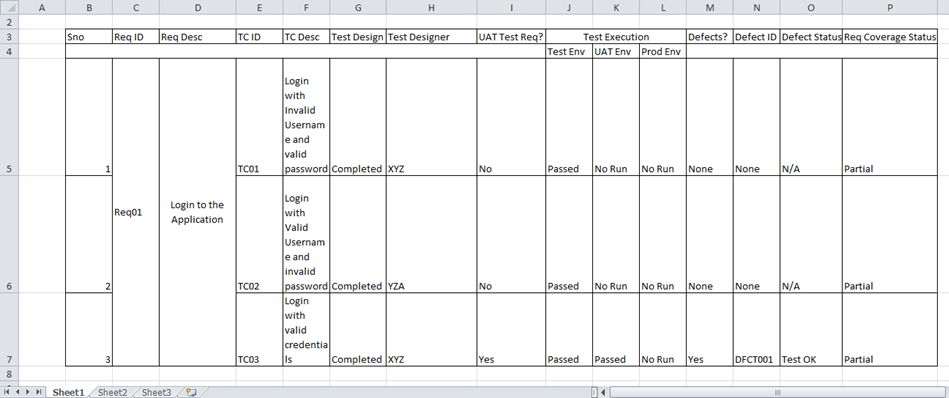
## **Which Parameters to include in Requirement Traceability Matrix?**

* Requirement ID
* Requirement Type and Description
* Test Cases with Status



Above is a sample requirement traceability matrix.

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



## **Goals of Traceability Matrix**

* It helps in tracing the documents that are developed during various phases of SDLC.
* It ensures that the software completely meets the customer's requirements.
* It helps in detecting the root cause of any bug.

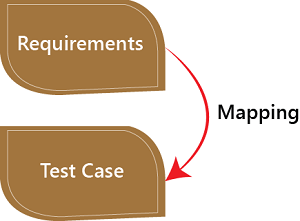
## **Types of Traceability Test Matrix**

The traceability matrix can be classified into three different types which are as follows:

* Forward traceability
* Backward or reverse traceability
* Bi-directional traceability

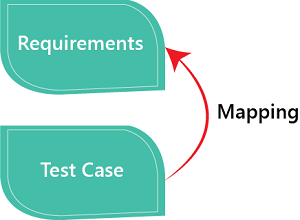
## **Forward traceability**

The forward traceability test matrix is used to ensure that every business's needs or requirements are executed correctly in the application and also tested rigorously. The main objective of this is to verify whether the product developments are going in the right direction. In this, the requirements are mapped into the forward direction to the test cases.



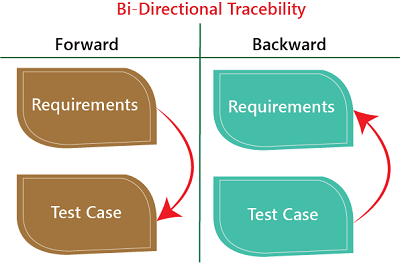
## **Backward or reverse traceability**

The reverse or backward traceability is used to check that we are not increasing the space of the product by enhancing the design elements, code, test other things which are not mentioned in the business needs. And the main objective of this that the existing project remains in the correct direction. In this, the requirements are mapped into the backward direction to the test cases.



### **Bi-directional traceability**

It is a combination of forwarding and backward traceability matrix, which is used to make sure that all the business needs are executed in the test cases. It also evaluates the modification in the requirement which is occurring due to the bugs in the application.



### **Advantage of RTM**

Following are the benefits of requirement traceability matrix:

* With the help of the RTM document, we can display the complete test execution and bugs status based on requirements.
* It is used to show the missing requirements or conflicts in documents.
* In this, we can ensure the complete test coverage, which means all the modules are tested.
* It will also consider the efforts of the testing teamwork towards reworking or reconsidering on the test cases.

**5.4.1 Test Scenario:-**

A **Test Scenario** is defined as any functionality that can be tested. It is also called Test Condition or Test Possibility. As a tester, you should put yourself in the end user’s shoes and figure out the real-world scenarios and use cases of the Application Under Test.

**Scenario Testing** in software testing is a method in which actual scenarios are used for testing the software application instead of test cases. The purpose of scenario testing is to test end to end scenarios for a specific complex problem of the software. Scenarios help in an easier way to test and evaluate end to end complicated problems.

The test scenario is a detailed document of test cases that cover end to end functionality of a software application in liner statements. The liner statement is considered as a scenario. The test scenario is a high-level classification of testable requirements. These requirements are grouped on the basis of the functionality of a module and obtained from the use cases.

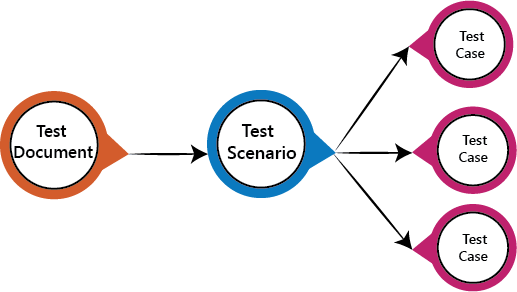
In the test scenario, there is a detailed testing process due to many associated test cases. Before performing the test scenario, the tester has to consider the test cases for each scenario.

In the test scenario, testers need to put themselves in the place of the user because they test the software application under the user's point of view. Preparation of scenarios is the most critical part, and it is necessary to seek advice or help from customers, stakeholders or developers to prepare the scenario.

**Note:**

The test scenarios can never be used for the text execution process because it does not consist of navigation steps and input.

These are the high-level documents that talks about all the possible combination or multiple ways or combinations of using the application and the primary purpose of the test scenarios are to understand the overall flow of the application.



## **How to write Test Scenarios**

As a tester, follow the following steps to create Test Scenarios-

* Read the requirement document such as BRS (Business Requirement Specification), SRS (System Requirement Specification) and FRS (Functional Requirement Specification) of the software which is under the test.
* Determine all technical aspects and objectives for each requirement.
* Find all the possible ways by which the user can operate the software.
* Ascertain all the possible scenario due to which system can be misused and also detect the users who can be hackers.
* After reading the requirement document and completion of the scheduled analysis make a list of various test scenarios to verify each function of the software.
* Once you listed all the possible test scenarios, create a traceability matrix to find out whether each and every requirement has a corresponding test scenario or not.
* Supervisor of the project reviews all scenarios. Later, they are evaluated by other stakeholders of the project.

## **Another Way 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 uses 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 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](https://www.guru99.com/traceability-matrix.html)is created to verify that each & every requirement has a corresponding Test Scenario

**Step 5:**The scenarios created are reviewed by your supervisor. Later, they are also reviewed by other Stakeholders in the project.

## **Features of Test Scenario**

The test scenario is a liner statement that guides testers for the testing sequence.

* Test scenario reduces the complexity and repetition of the product.
* Test scenario means talking and thinking about tests in detail but write them in liner statements.
* It is a thread of operations.
* Test scenario becomes more important when the tester does not have enough time to write test cases, and team members agree with a detailed liner scenario.
* The test scenario is a time saver activity.
* It provides easy maintenance because the addition and modification of test scenarios are easy and independent.

**Note:**

Some rules have to be followed when we were writing test scenarios:-

* Always list down the most commonly used feature and module by the users.
* We always start the scenarios by picking module by module so that a proper sequence is followed as well as we don't miss out on any module level.
* Generally, scenarios are module level.
* Delete scenarios should always be the last option else, and we will waste lots of time in creating the data once again.
* It should be written in a simple language.
* Every scenario should be written in one line or a maximum of two lines and not in the paragraphs.
* Every scenario should consist of Dos and checks.

## **Why create Test Scenarios?**

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.

## **Example of Test scenarios :-**

Here we are taking the **Gmail application** and writing test scenarios for different modules which are most commonly used such as **Login, Compose, Inbox, and Trash**

### **Test scenarios on the Login module**

* Enter the valid login details (Username, password), and check that the home page is displayed.
* Enter the invalid Username and password and check for the home page.
* Leave Username and password blank, and check for the error message displayed.
* Enter the valid Login, and click on the cancel, and check for the fields reset.
* Enter invalid Login, more than three times, and check that account blocked.
* Enter valid Login, and check that the **Username** is displayed on the home screen.

### **Test scenarios on Compose module**

* Checks that all users can enter email ides in the **To, Cc, and Bcc**.
* Check that the entire user can enter various email ids in To, Cc, and Bcc.
* Compose a mail, send it, and check for the confirmation message.
* Compose a mail, send it, and check in the sent item of the sender and the inbox.
* Compose a mail, send it, and check for invalid and valid email id (valid format), check the mail in sender inbox.
* Compose main, discard, and then check for conformation message and check-in draft.
* Compose mail click on save as draft and check for the confirmation message
* Compose mail click on close and check for conformation save as drafts.

### **Test scenarios on Inbox module**

* Click on the inbox, and verify all received mail are displayed and highlighted in the inbox.
* Check that a latest received mail has been displayed to the sender email id correctly.
* Select the mail, reply and forward send it; check in the sent item of sender and inbox of the receiver.
* Check for any attached attachments to the mail that are downloaded or not.
* Check that attachment is scanned correctly for any viruses before download.
* Select the mail, reply and forward save as draft, and check for the confirmation message and checks in the Draft section.
* Check all the emails are marked as read are not highlighted.
* Check all mail recipients in **Cc** are visible to all users.
* Checks all email recipients in **Bcc** are not visible to the users.
* Select mail, delete it, and then check in the **Trash** section.

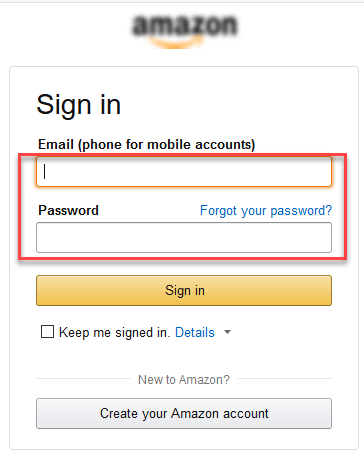
### **Test scenario on Trash module**

* Open trash, check all deleted mail present.
* Restore mail from Trash; check-in the corresponding module.
* Select mail from trash, delete it, and check mail is permanently deleted.

### **Another Example : Test Scenario for ecommerce Application**

For an eCommerce Application, a few test scenarios would be

**Test Scenario 1:**Check the Login Functionality



In order to help you understand the difference Test Scenario and Test Cases, specific test cases for this Test Scenario would be

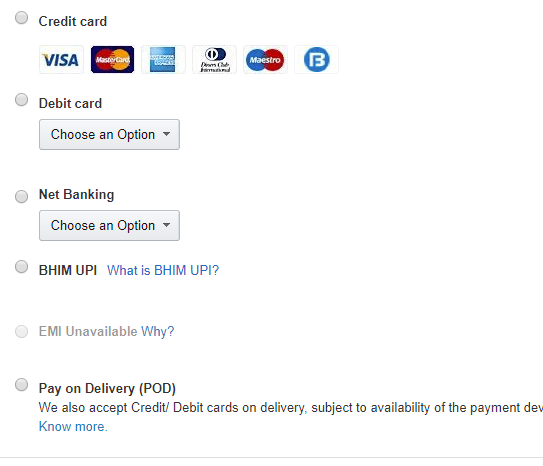
1. Check system behaviour when valid email id and password is entered.
2. Check system behaviour when invalid email id and valid password is entered.
3. Check system behaviour when valid email id and invalid password is entered.
4. Check system behaviour when invalid email id and invalid password is entered.
5. Check system behaviour when email id and password are left blank and Sign in entered.
6. Check Forgot your password is working as expected
7. Check system behaviour when valid/invalid phone number and password is entered.
8. Check system behaviour when “Keep me signed” is checked

**Test Scenario 2: Check the Search Functionality**

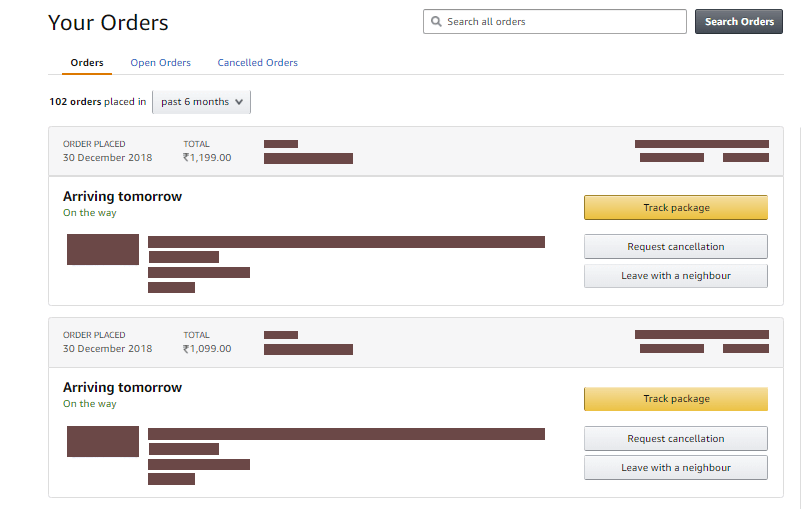
test scenario

**Test Scenario 3: Check the Product Description Page**

**Test Scenario 4: Check the Payments Functionality**



**Test Scenario 5: Check the Order History**



**Apart from these 5 scenarios here is the list of all other scenarios**

* Check Home Page behaviour for returning customers
* Check Category/Product Pages
* Check Customer Service/Contact Pages
* Check Daily Deals pages

**5.4.2 Test Suite :-**

It is a collection of [test cases](https://en.wikipedia.org/wiki/Test_case) that are intended to be used to test a software program to show that it has some specified set of behaviours. A test suite often contains detailed instructions or goals for each collection of test cases and information on the system configuration to be used during testing. A group of test cases may also contain prerequisite states or steps, and descriptions of the following tests.

A test suite allows you to categorize test cases in such a way that they match your planning and analysis needs. Do you run functional and performance tests? Create two suites and label them accordingly.

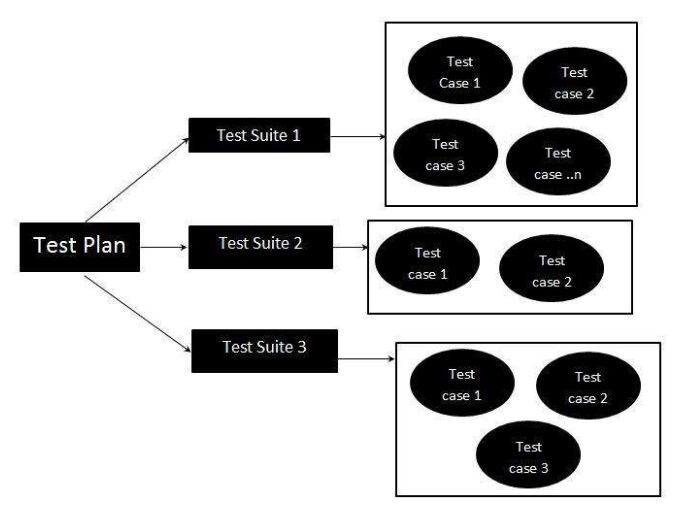
Test suites are used to group similar test cases together. A system might have a smoke test suite that consists only of [smoke tests](https://en.wikipedia.org/wiki/Smoke_testing_(software)) or a test suite for some specific functionality in the system. It may also contain all tests and signify if a test should be used as a smoke test or for some specific functionality.

Test suite is a container that has a set of tests which helps testers in executing and reporting the test execution status. It can take any of the three states namely Active, In progress and completed.

A Test case can be added to multiple test suites and test plans. After creating a test plan, test suites are created which in turn can have any number of tests.

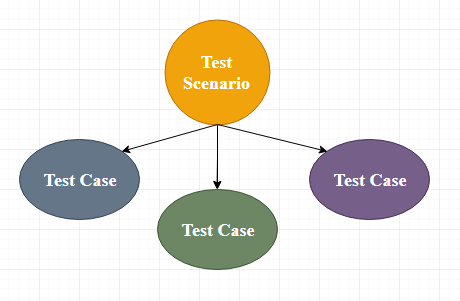
Test suites are created based on the cycle or based on the scope. It can contain any type of tests, viz - functional or Non-Functional.

## **Test Suite - Diagram:**

****

* **5.4 Test Cases – Definition, Test Case Designing**

The test case is defined as a group of conditions under which a tester determines whether a software application is working as per the customer's requirements or not. Test case designing includes preconditions, case name, input conditions, and expected result. A test case is a first level action and derived from test scenarios.



It is an in-details document that contains all possible inputs (positive as well as negative) and the navigation steps, which are used for the test execution process. Writing of test cases is a one-time attempt that can be used in the future at the time of regression testing.

Test case gives detailed information about testing strategy, testing process, preconditions, and expected output. These are executed during the testing process to check whether the software application is performing the task for that it was developed or not.

Test case helps the tester in defect reporting by linking defect with test case ID. Detailed test case documentation works as a full proof guard for the testing team because if developer missed something, then it can be caught during execution of these full-proof test cases.

## **Why we write the test cases?**

We will write the test for the following reasons:

* **To require consistency in the test case execution**
* **To make sure a better test coverage**
* **It depends on the process rather than on a person**
* **To avoid training for every new test engineer on the product**

**To require consistency in the test case execution:** we will see the test case and start testing the application.

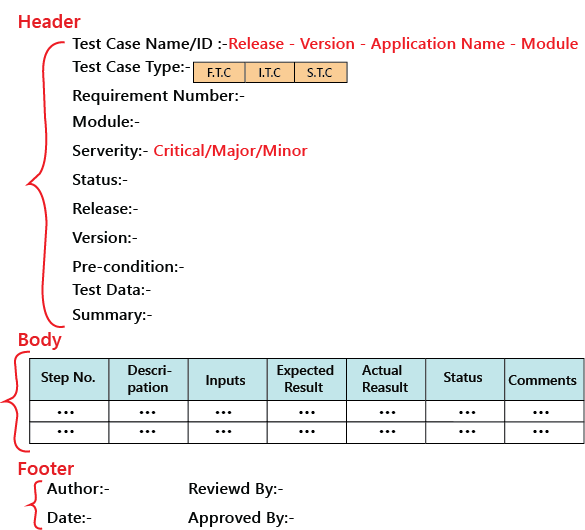
**To make sure a better test coverage:** for this, we should cover all possible scenarios and document it, so that we need not remember all the scenarios again and again.

**It depends on the process rather than on a person:** A test engineer has tested an application during the first release, second release, and left the company at the time of third release. As the test engineer understood a module and tested the application thoroughly by deriving many values. If the person is not there for the third release, it becomes difficult for the new person. Hence all the derived values are documented so that it can be used in the future.

**To avoid giving training for every new test engineer on the product:** When the test engineer leaves, he/she leaves with a lot of knowledge and scenarios. Those scenarios should be documented so that the new test engineer can test with the given scenarios and also can write the new scenarios.

## **Test case template**

The primary purpose of writing a test case is to achieve the efficiency of the application.



The **actual result** is written after the test case execution, and most of the time, it would be same as the **expected result**. But if the test step will fail, it will be different. So, the actual result field can be skipped, and in the **Comments** section, we can write about the bugs.

And also, the **Input field** can be removed, and this information can be added to the **Description field**.

**More Description :**

1. A test case tests the functionality to be specific object test.

2. A test case is a description of what to be tested, what data to be given and what action to done to check actual result against the expected result.

3. A test cases list the specific items that will be tested and describe the detailed steps that will be followed to verify the s/w.

4. It is the document which includes steps description of i/p, o/p conditions with test data along with expected result.

**Test Case Designing:**

* Test cases involve the set of steps, conditions and inputs which can be used while performing the testing tasks.
* The main intent of this activity is to ensure whether the Software Passes or Fails in terms of its functionality and other aspects.
* There are many types of test cases like: functional, negative, error, logical test cases, physical test cases, UI test cases etc.
* Furthermore test cases are written to keep track of testing coverage of Software.
* Generally, there is no formal template which is used during the test case writing, however following are the ***main components*** which are always available and included in every ***test case:***
  + Test case ID.
  + Product Module.
  + Product version.
  + Revision history.
  + Purpose.
  + Assumptions.
  + Pre-Conditions.
  + Steps.
  + Expected Outcome.
  + Actual Outcome.
  + Post Conditions.

|  |  |  |  |
| --- | --- | --- | --- |
| **Test case ID:**  **Product Module:** | <TC ID>  <Home Page> | **Test Engineer:**  **Test Date:** | < Test Engineer >  08-07-2014 |
| **Product version:** |  | **Testing Cycle:** | 1 |
| **Revision history:** |  | **Status:** |  |
| **Purpose:** | <Purpose> | | |
| **Assumptions:** | <Assumptions> | | |
| **Pre-Conditions:** | <Pre-Conditions> | | |
| **Steps to reproduce** | <Steps to reproduce> | | |
| **Expected Result:** | <Expected Outcome> | | |
| **Actual Result:** | < Actual Outcome > | | |
| **Post Conditions:** | <Purpose> | | |

* **5.5 Configuration Management- Configuration Management support for Testing**

**Software Configuration Management(SCM)** is a process to systematically manage, organize, and control the changes in the documents, codes, and other entities during the Software Development Life Cycle. The primary goal is to increase productivity with minimal mistakes. SCM is part of cross-disciplinary field of configuration management and it can accurately determine who made which revision.

When we develop software, the product (software) undergoes many changes in their maintenance phase; we need to handle these changes effectively.

Several individuals (programs) works together to achieve these common goals. This individual produces several work product (SC Items) e.g., Intermediate version of modules or test data used during debugging, parts of the final product.

The elements that comprise all information produced as a part of the software process are collectively called a software configuration.

As software development progresses, the number of Software Configuration elements (SCI's) grow rapidly.

***These are handled and controlled by SCM. This is where we require software configuration management*.**

A configuration of the product refers not only to the product's constituent but also to a particular version of the component.

Therefore, SCM is the discipline which

* Identify change
* Monitor and control change
* Ensure the proper implementation of change made to the item.
* Auditing and reporting on the change made.

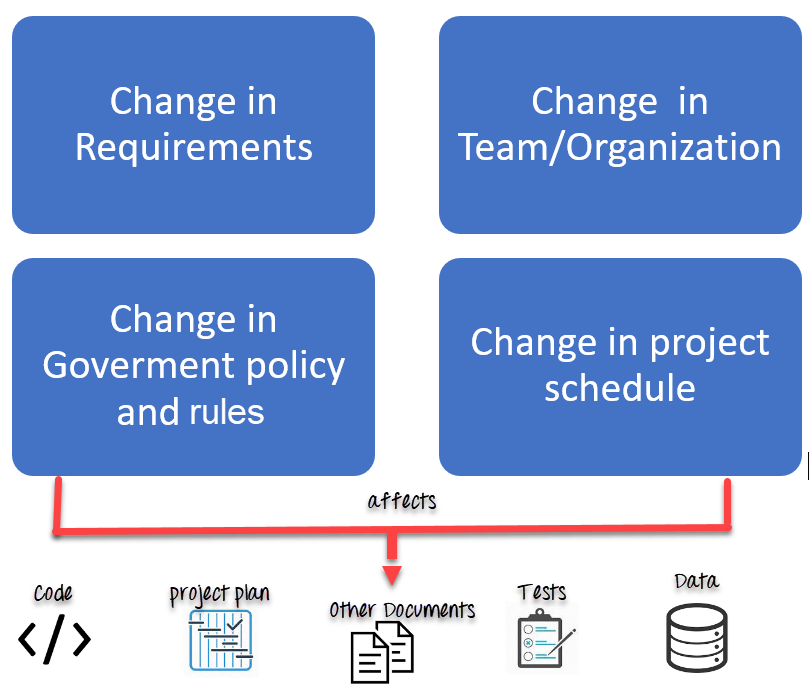
Configuration Management (CM) is a technic of identifying, organizing, and controlling modification to software being built by a programming team.

**The objective is to maximize productivity by minimizing mistakes (errors).**

CM is used to essential due to the inventory management, library management, and updation management of the items essential for the project.

## **Why do we need Configuration management?**

* There are multiple people working on software which is continually updating
* It may be a case where multiple version, branches, authors are involved in a software config project, and the team is geographically distributed and works concurrently
* Changes in user requirement, policy, budget, schedule need to be accommodated.
* Software should able to run on various machines and Operating Systems
* Helps to develop coordination among stakeholders
* SCM process is also beneficial to control the costs involved in making changes to a system



## **Tasks in SCM process**

* Configuration Identification
* Baselines
* Change Control
* Configuration Status Accounting
* Configuration Audits and Reviews

## **Configuration Identification:**

Configuration identification is a method of determining the scope of the software system. With the help of this step, you can manage or control something even if you don’t know what it is. It is a description that contains the CSCI type (Computer Software Configuration Item), a project identifier and version information.

## **Activities during this process:**

Identification of configuration Items like source code modules, test case, and requirements specification.

Identification of each CSCI in the SCM repository, by using an object-oriented approach

The process starts with basic objects which are grouped into aggregate objects. Details of what, why, when and by whom changes in the test are made

Every object has its own features that identify its name that is explicit to all other objects

List of resources required such as the document, the file, tools, etc.

## **Example:**

Instead of naming a File login.php its should be named login\_v1.2.php where v1.2 stands for the version number of the file

Instead of naming folder “Code” it should be named “Code\_D” where D represents code should be backed up daily.

## **Baseline:**

A baseline is a formally accepted version of a software configuration item. It is designated and fixed at a specific time while conducting the SCM process. It can only be changed through formal change control procedures.

## **Activities during this process:**

* Facilitate construction of various versions of an application
* Defining and determining mechanisms for managing various versions of these work products
* The functional baseline corresponds to the reviewed system requirements
* Widely used baselines include functional, developmental, and product baselines

In simple words, baseline means ready for release.

## **Change Control:**

Change control is a procedural method which ensures quality and consistency when changes are made in the configuration object. In this step, the change request is submitted to software configuration manager.

## **Activities during this process:**

* Control ad-hoc change to build stable software development environment. Changes are committed to the repository
* The request will be checked based on the technical merit, possible side effects and overall impact on other configuration objects.
* It manages changes and making configuration items available during the software lifecycle

## **Configuration Status Accounting:**

* Configuration status accounting tracks each release during the SCM process. This stage involves tracking what each version has and the changes that lead to this version.

## **Activities during this process:**

* Keeps a record of all the changes made to the previous baseline to reach a new baseline
* Identify all items to define the software configuration
* Monitor status of change requests
* Complete listing of all changes since the last baseline
* Allows tracking of progress to next baseline
* Allows to check previous releases/versions to be extracted for testing

## **Configuration Audits and Reviews:**

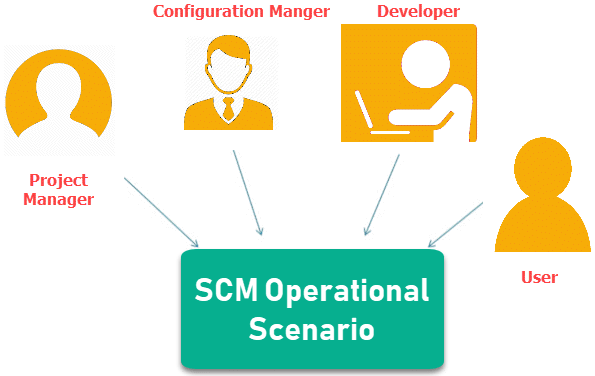
Software Configuration audits verify that all the software product satisfies the baseline needs. It ensures that what is built is what is delivered.

## **Activities during this process:**

* Configuration auditing is conducted by auditors by checking that defined processes are being followed and ensuring that the SCM goals are satisfied.
* To verify compliance with configuration control standards. auditing and reporting the changes made
* SCM audits also ensure that traceability is maintained during the process.
* Ensures that changes made to a baseline comply with the configuration status reports
* Validation of completeness and consistency

## **Participant of SCM process**:

Following are the key participants in SCM



**1. Configuration Manager**

* Configuration Manager is the head who is Responsible for identifying configuration items.
* CM ensures team follows the SCM process
* He/She needs to approve or reject change requests

**2. Developer**

* The developer needs to change the code as per standard development activities or change requests. He is responsible for maintaining configuration of code.
* The developer should check the changes and resolves conflicts

**3. Auditor**

* The auditor is responsible for SCM audits and reviews.
* Need to ensure the consistency and completeness of release.

**4. Project Manager:**

* Ensure that the product is developed within a certain time frame
* Monitors the progress of development and recognizes issues in the SCM process
* Generate reports about the status of the software system
* Make sure that processes and policies are followed for creating, changing, and testing

**5. User**

* The end user should understand the key SCM terms to ensure he has the latest version of the software
* **Configuration Management Support for Testing :-**

If configurations are not managed well, a necessary update of a test script or test data for a newer software version could create an incapability to cover previous software versions in software testing – e.g. after a defect occurred in the live software version, retesting or retracing the same could become impossible or expensive in the available test environment.

Therefore it is important to consider the software deployment cycle in the test plan not only regarding software test cycles but also regarding software test configurations. Test items and testware have to be preserved as long as a certain software version is available. Usually, one ore more reference and release test environments are being provided to maintain a basic capability. Nevertheless, configurations of the same are not being guaranteed to provide the correct comprehensive configuration to maintain a required level of representation and reliability. Test planning should consider the mismatches in configurations that might apply in the different test environments and describe actions to compensate them, if they are inevitable in terms of adjusting configurations itself.

Test data could sometimes be left out in consideration. Most common is the insufficient coverage of the third precondition, mentioned above as traceability of test items and testware items towards referred documentation. Especially in agile environments, where one of the statements of the manifest is “Working software over comprehensive documentation”, fulfilling the corresponding precondition might sound difficult to achieve.

* **5.6 Risk and Testing- Project Risk & Product Risk :-**

"Tomorrow problems are today's risk." Hence, a clear definition of a "risk" is a problem that could cause some loss or threaten the progress of the project, but which has not happened yet.

These potential issues might harm cost, schedule or technical success of the project and the quality of our software device, or project team morale.

Risk Management is the system of identifying addressing and eliminating these problems before they can damage the project.

We need to differentiate risks, as potential issues, from the current problems of the project.

Risk management is the process of identifying, assessing, and prioritizing the risks to minimize, monitor, and control the probability of unfortunate events.

Risk is the probability of occurrence of an undesirable event.

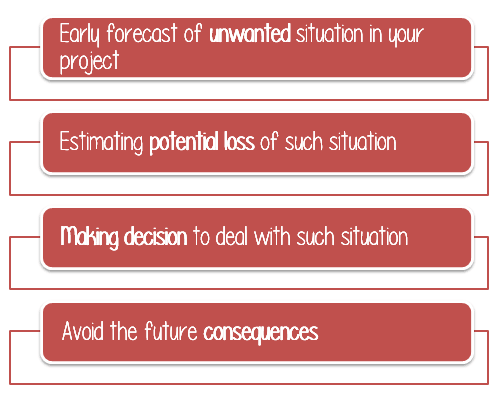
Risk Analysis in Software Engineering is the process of analysing the risks associated with your Project.

## **Risk Management Process:**

Risk Management process can be easily understood with use of the following workflow:



## **Risk management helps you in –**



## **Principle of Risk Management**

1. **Global Perspective:** In this, we review the bigger system description, design, and implementation. We look at the chance and the impact the risk is going to have.
2. **Take a forward-looking view:** Consider the threat which may appear in the future and create future plans for directing the next events.
3. **Open Communication:** This is to allow the free flow of communications between the client and the team members so that they have certainty about the risks.
4. **Integrated management:** In this method risk management is made an integral part of project management.
5. **Continuous process:** In this phase, the risks are tracked continuously throughout the risk management paradigm.

## **How to Perform Risk ANALYSIS?**

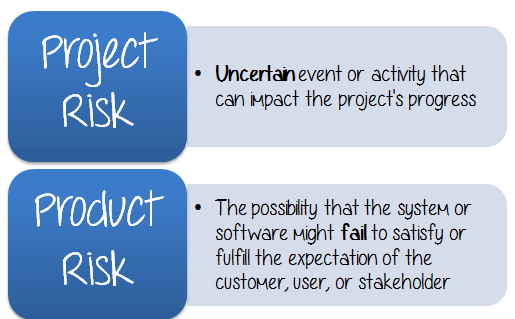
It’s a 3-Step process

1. Identify the Risks
2. Analyze Impact of each Identified Risk
3. Take counter measures for the identified & Analyzed risk



## **Identify Risk**

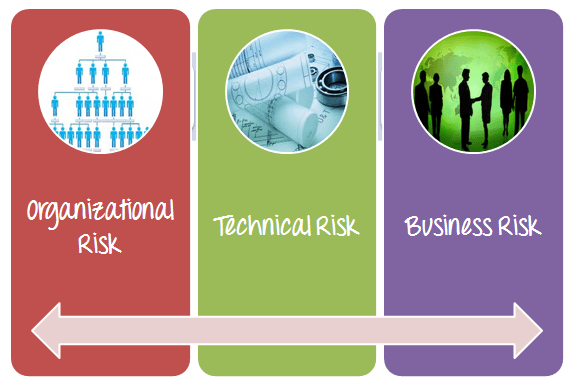
Risk can be identified and classified into 2 types in software product



## **Project Risk**

Project risk can be defined as an **uncertain** event or activity that can impact the project’s progress. The impact has a **positive** or **negative** effect on the prospects of achieving project objectives.

There are primarily 3 categories of Project Risks



## **Technical Risk**

Technical Risk is the probability of loss incurred during the execution of a technical process such as untested engineering, wrong testing procedure…etc. Here is an example of technical risk

Your task in this project is testing a banking website. You have to set up proper test environments which mirror real business environments. If the[Test Environment](https://www.guru99.com/test-environment-software-testing.html)is not setup properly, the product will be **not**be tested correctly and many **defects**will not be detected.

## **Business Risk**

The risk involves an **external** entity. It is the risk which may come from your company, your customer but **not** from your project.

In such case, the Test Manager has to find out the solutions to deal with the risk such as:

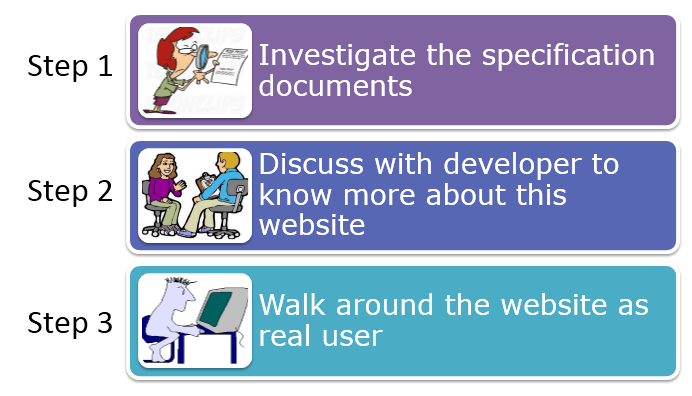
* Set **priority** for the testing phases, focus on testing the main features of website
* **Utilize**a testing tool to increase the productivity of testing
* Apply **process improvement** to reduce the management effort.

## **Product Risk**

**Product risk**is the possibility that the system or software might fail to satisfy or fulfill the expectation of the customer, user, or stakeholder. This Risk in Test Plan is related to the **functionality** of the product such as Performance Issues, Security Issues, Crash Scenarios, etc.

**Following are examples of a few product risks –**

* The software skips some **key** function that the customers specified in the users’  
  requirement
* The software is **unreliable** and frequently **fails** to work.
* Software fail in ways that cause financial or other damage to a user or the company that uses the software.
* The software has problems related to a particular quality characteristic such as security, reliability, usability, maintainability or performance.



## **Step 2) Analyze the impact of the risk occurring**

In the previous topic, we already identified the risks which may hamper your project. Here is the list of risks identified:

* You may not have enough **human resource** to finish the project on the deadline
* The Testing **environment**may not be setup properly like real business environment.
* Your project **budget**may cut by half because of business situation
* This website may **lack** security functions

Next, you should analyze these risks.

Each risk should be classified on the basis of following two parameters

* The **probability** of occurrence
* The **impact** on the project

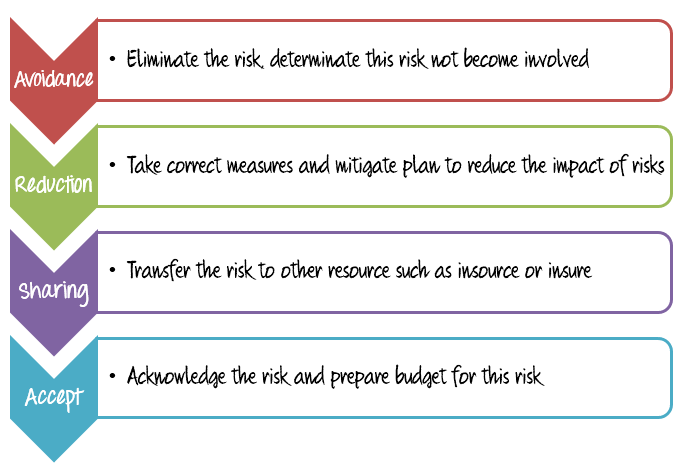
## **Step 3) Take COUNTERMEASURES to mitigate the risk**

This activity is divided into 3 parts



## **Risk response**

The project manager needs to choose strategies that will reduce the risk to minimal. Project managers can choose between the following four risk response strategies



## **Register Risk**

All the risk must be recorded, documented and acknowledged by project managers, stakeholder and the project member. The risk register should be freely accessible to all the members of the project team.

## **Monitor and Control Risk**

Risks can be monitored on a continuous basis to check if any changes are made. New risk can be identified through the constant monitoring and assessing mechanisms.

* **5.7 Incident/ Defect Management :-**
* **5.7.1 Defect Life Cycle :-**

**Defect Life Cycle** or Bug Life Cycle in software testing is the specific set of states that defect or bug goes through in its entire life. The purpose of Defect life cycle is to easily coordinate and communicate current status of defect which changes to various assignees and make the defect fixing process systematic and efficient.

Defect life cycle, also known as Bug Life cycle is the journey of a defect cycle, which a defect goes through during its lifetime. It varies from organization to organization and also from project to project as it is governed by the software testing process and also depends upon the tools used.

## 

* **New** - Potential defect that is raised and yet to be validated.
* **Assigned** - Assigned against a development team to address it but not yet resolved.
* **Active** - The Defect is being addressed by the developer and investigation is under progress. At this stage there are two possible outcomes; viz - Deferred or Rejected.
* **Test** - The Defect is fixed and ready for testing.
* **Verified** - The Defect that is retested and the test has been verified by QA.
* **Closed** - The final state of the defect that can be closed after the QA retesting or can be closed if the defect is duplicate or considered as NOT a defect.
* **Reopened** - When the defect is NOT fixed, QA reopens/reactivates the defect.
* **Deferred** - When a defect cannot be addressed in that particular cycle it is deferred to future release.

## https://www.guru99.com/images/1-2015/012715_0802_BugLifeCycl1.png

## **Defect Status**

Defect Status or Bug Status in defect life cycle is the present state from which the defect or a bug is currently undergoing. The goal of defect status is to precisely convey the current state or progress of a defect or bug in order to better track and understand the actual progress of the defect life cycle.

The number of states that a defect goes through varies from project to project. Below lifecycle diagram, covers all possible states

* **New:** When a new defect is logged and posted for the first time. It is assigned a status as NEW.
* **Assigned:** Once the bug is posted by the tester, the lead of the tester approves the bug and assigns the bug to the developer team
* **Open**: The developer starts analyzing and works on the defect fix
* **Fixed**: When a developer makes a necessary code change and verifies the change, he or she can make bug status as “Fixed.”
* **Pending retest**: Once the defect is fixed the developer gives a particular code for retesting the code to the tester. Since the software testing remains pending from the testers end, the status assigned is “pending retest.”
* **Retest**: Tester does the retesting of the code at this stage to check whether the defect is fixed by the developer or not and changes the status to “Re-test.”
* **Verified**: The tester re-tests the bug after it got fixed by the developer. If there is no bug detected in the software, then the bug is fixed and the status assigned is “verified.”
* **Reopen**: If the bug persists even after the developer has fixed the bug, the tester changes the status to “reopened”. Once again the bug goes through the life cycle.
* **Closed**: If the bug is no longer exists then tester assigns the status “Closed.”
* Duplicate: If the defect is repeated twice or the defect corresponds to the same concept of the bug, the status is changed to “duplicate.”
* **Rejected**: If the developer feels the defect is not a genuine defect then it changes the defect to “rejected.”
* **Deferred**: If the present bug is not of a prime priority and if it is expected to get fixed in the next release, then status “Deferred” is assigned to such bugs
* **Not a bug**: If it does not affect the functionality of the application then the status assigned to a bug is “Not a bug”.



* **5.7.2. Defect/ Incident Report:-**

A defect in a software product is also known as a bug, error or fault which makes the software produce an unexpected result as per the software requirements. For example; incorrect data, system hangs, unexpected errors, missing or incorrect requirements.

A **Defect in Software Testing** is a variation or deviation of the software application from end user’s requirements or original business requirements. A software defect is an error in coding which causes incorrect or unexpected results from a software program which does not meet actual requirements.

**Defect Report:**

A defect report is a document that has concise details about what defects are identified, what action steps make the defects show up, and what are the expected results instead of the application showing error (defect) while taking particular step by step actions.

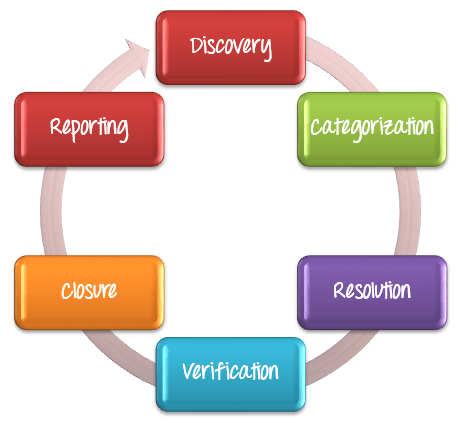
A Bug Report in Software Testing is a detailed document about bugs found in the software application. Bug report contains each detail about bugs like description, date when bug was found, name of tester who found it, name of developer who fixed it, etc. Bug report helps to identify similar bugs in future so it can be avoided.

While reporting the bug to developer, your Bug Report should contain the following information

* **Defect\_ID** – Unique identification number for the defect.
* **Defect Description** – Detailed description of the Defect including information about the module in which Defect was found.
* **Version** – Version of the application in which defect was found.
* **Steps** – Detailed steps along with screenshots with which the developer can reproduce the defects.
* **Date Raised** – Date when the defect is raised
* **Reference**– where in you Provide reference to the documents like . requirements, design, architecture or maybe even screenshots of the error to help understand the defect
* **Detected By** – Name/ID of the tester who raised the defect
* **Status** – Status of the defect
* **Fixed by** – Name/ID of the developer who fixed it
* **Date Closed** – Date when the defect is closed
* **Severity-** which describes the impact of the defect on the application
* **Priority-**which is related to defect fixing urgency. Severity Priority could be High/Medium/Low based on the impact urgency at which the defect should be fixed respectively

## **What is Defect Management Process?**

Defect Management is a systematic process to identify and fix bugs. A defect management cycle contains the following stages 1) Discovery of Defect, 2) Defect Categorization 3) Fixing of Defect by developers 4) Verification by Testers, 5) Defect Closure 6) Defect Reports at the end of project



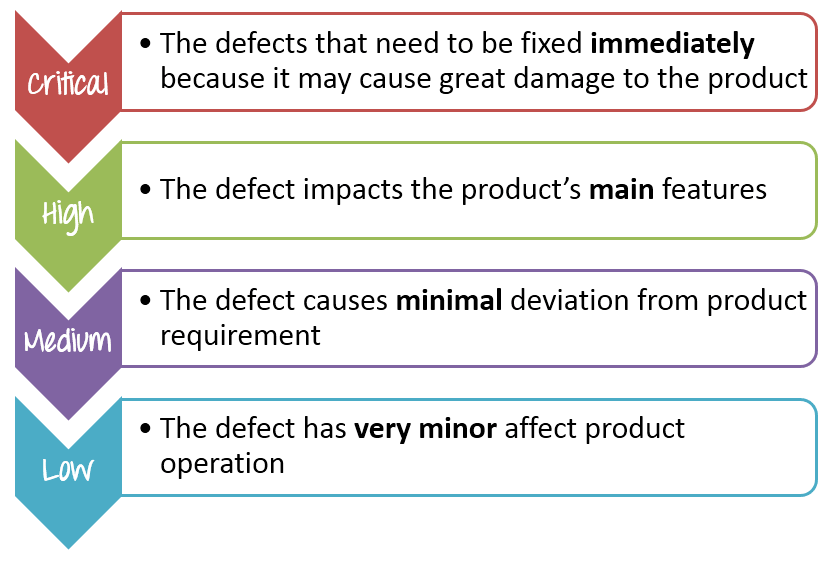
## **Discovery**

In the discovery phase, the project teams have to discover as **many** defects as **possible,**before the end customer can discover it. A defect is said to be discovered and change to status **accepted** when it is acknowledged and accepted by the developers



## **Categorization**

Defect categorization help the software developers to prioritize their tasks. That means that this kind of priority helps the developers in fixing those defects first that are highly crucial.



## **Defect Resolution**

**Defect Resolution** in software testing is a step by step process of fixing the defects. Defect resolution process starts with assigning defects to developers, then developers schedule the defect to be fixed as per priority, then defects are fixed and finally developers send a report of resolution to the test manager. This process helps to fix and track defects easily.

You can follow the following steps to fix the defect.



* **Assignment**: Assigned to a developer or other technician to fix, and changed the status to **Responding**.
* **Schedule fixing**: The developer side take charge in this phase. They will create a schedule to fix these defects, depend on the defect priority.
* **Fix the defect**: While the development team is fixing the defects, the Test Manager tracks the process of fixing defect compare to the above schedule.
* **Report the resolution**: Get a report of the resolution from developers when defects are fixed.

## **Verification**

After the development team **fixed** and **reported** the defect, the testing team **verifies** that the defects are actually resolved.

For example, in the above scenario, when the development team reported that they already fixed 61 defects, your team would test again to verify these defects were actually fixed or not.

## **Closure**

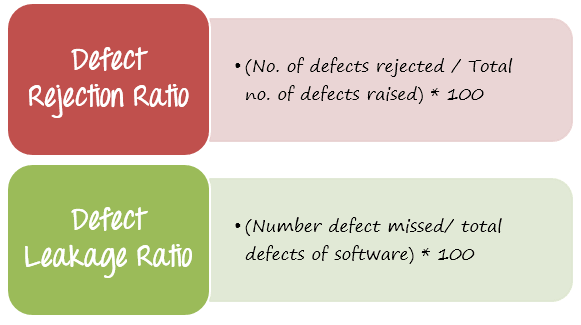
Once a defect has been resolved and verified, the defect is changed status as **closed**. If not, you have send a notice to the development to check the defect again.

## **Defect Reporting**

**Defect Reporting** in software testing is a process in which test managers prepare and send the defect report to the management team for feedback on defect management process and defects’ status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail.

## **How to measure and evaluate the quality of the test execution?**

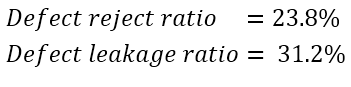
This is a question which every Test Manager wants to know. There are 2 parameters which you can consider as following



In the above scenario, you can calculate the defection rejection ratio (DRR) is 20/84 = 0.238 (23.8 %).

Another example, supposed the Bank website has total 64 defects, but your testing team only detect 44 defects i.e. they missed 20 defects. Therefore, you can calculate the defect leakage ratio (DLR) is 20/64 = 0.312 (31.2 %).

Conclusion, the quality of test execution is evaluated via following two parameters



The smaller value of DRR and DLR is, the better quality of test execution is. What is the ratio range which is acceptable? This range could be defined and accepted base in the project target or you may refer the metrics of similar projects.

In this project, the recommended value of acceptable ratio is 5 ~ 10%. It means the quality of test execution is low. You should find countermeasure to reduce these ratios such as

* **Improve**the testing skills of member.
* **Spend more time** for testing execution, especially for reviewing the test execution results.