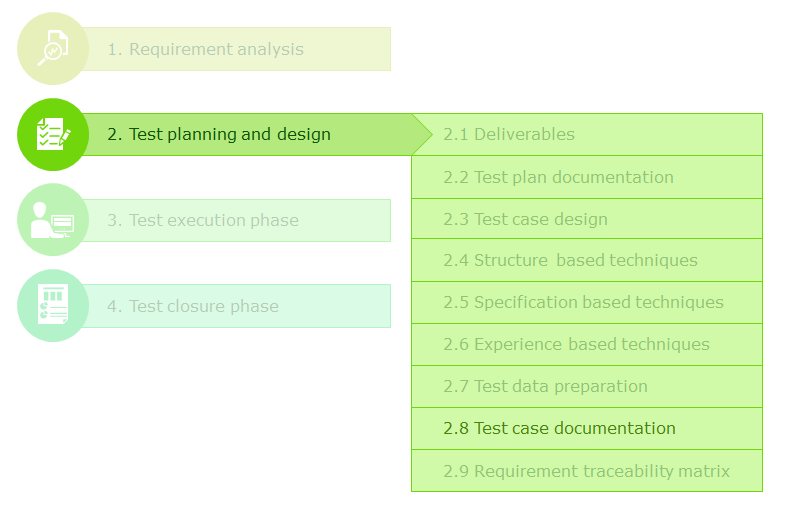
2.8 Test Case Documentation



Test case design is performed much before the actual execution, depending on the type of testing (as per the V-Model). Sometimes the gap between test case preparation and test execution can even be several months.

Once the test cases are prepared using the test case design optimization techniques described earlier, they are supposed to be documented in a formal and elaborate document. This helps in

* Not missing out any tests during execution
* Reducing knowledge and communication related rework if tests are going to be designed by one tester and executed by another.
* Providing a base document for reference, review, planning and delegation of testing related SQC activities.

Apart from documenting the steps for executing tests and the expected results, certain additional information is also documented for effectively planning and executing the tests. In the next page let's look all such information can be documented.

**Test Case Template**

Given below are the information that is usually captured in the Test Case Document. The number of fields and the format of documenting may vary from one project to another. However it is still important to understand the significance of each and every field.

**Requirement ID** : This is an alpha-numeric code used to uniquely identify the corresponding requirement(s) for which the test case is being written. It may be directly taken from the requirements document, if available, or formatted as per the conventions followed in the team. This helps, during reviews and checks, to ensure that every requirement has been addressed adequately

**Requirement Description** : A brief description of the requirement so that anybody who is going to review or execute the test case need not refer to the requirement document frequently to understand it.

**Test Case ID** : This is an alpha-numeric code used to uniquely identify the test case. This helps, along with the 'Requirement ID', to simplify the representation of relationship between test cases and requirements. Again, it helps during reviews, checks and test execution.

**Test Case Description** : One requirement might have to be tested with multiple test cases. It is imperative to briefly describe what the objective of every test case is. This helps the test executor in understanding the necessity of every test case.

**Precondition**: This field is used to describe the starting state in which the application should be, before the test case can be executed, along with the steps/actions required to achieve that state.

**Test Data** : This field states any specific input data that needs to be provided to the application for executing the test case successfully.

**Type of Check** : This field is used if there are any internal guidelines in the project to categorize the test cases for management, delegation, tracking, application classification or any such purposes. E.g., type of tests, types of UI checks, types of performance checks, etc.

**Test Step Number**: Each test cases is recorded as a sequence of actions to be performed by the tester. Each such action is called a test step. The test step number is used to sequence the test steps of a test case. For an experienced tester/lead/reviewer it is possible to deduce the size, complexity (or both) of a test case based on number of test steps. This in turn helps in effort estimation, planning and delegation, whenever required.

**Test Step Description**: This field is used to describe the action that needs to be performed to execute the corresponding test step.

**Expected Result** : This field is used to describe the expected outcome of the corresponding test step. If, during execution, the actual outcome does not match the expected result stated here, then a defect is logged accordingly.

**Actual Result** : This field is used to record the actual outcome of a test step during execution. During test preparation it is left empty. The information captured might be recorded as a simple PASS or FAIL status or an elaborated description of the system behavior, especially if the step is failing.

**Tester Name** : To capture the name of the author/tester of a test case. If the test author and actual tester are going to be two different persons, then two fields can be used. This is used for delegation, tracking and audit purposes.

**Creation Date** : Date of preparation of the test case in question for the purpose of tracking and auditing.

**Problem Statement:**

**Objective**

To practice how to develop test cases from requirements

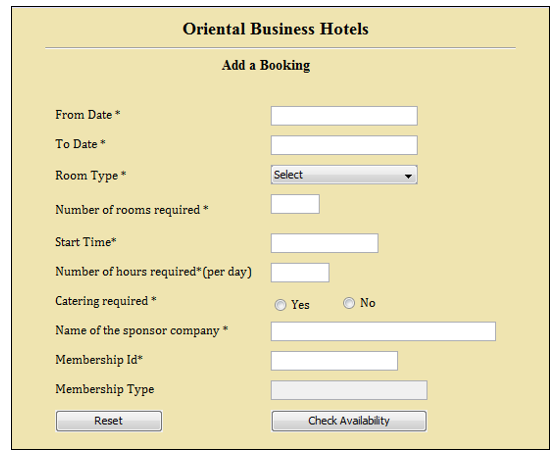
**Problem statement**

Derive test cases for the requirement for the software module given below with the help of the test case template sample provided earlier.

**Requirements**

Oriental Business Hotels website enables the users who have membership with the hotel group to book Banquet Halls, Conference halls, Media rooms etc. across all the locations in India.

**Add a Booking Page - UI Lookup for reference**



* Here, the user has to input appropriate data into the fields in the page and click on ‘Check availability’ button. On successful submission the user gets the message **“<<Room type>> is available for the date <<From Date>> to <<To Date from <<start time>> to <<end time>>.”**

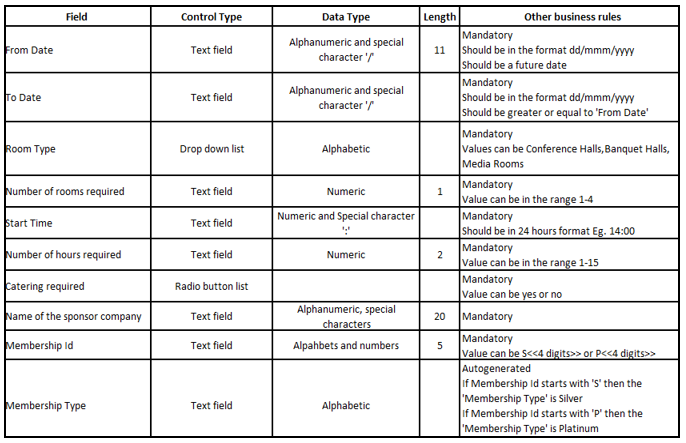
       End time = Start time + Number of hours required (per day)

       The inputs entered by the user should meet the business rules mentioned in Table below, else appropriate error messages are displayed on the same screen.

* On clicking the ‘Reset’ button, all fields are set to the default value.

Business Rules:

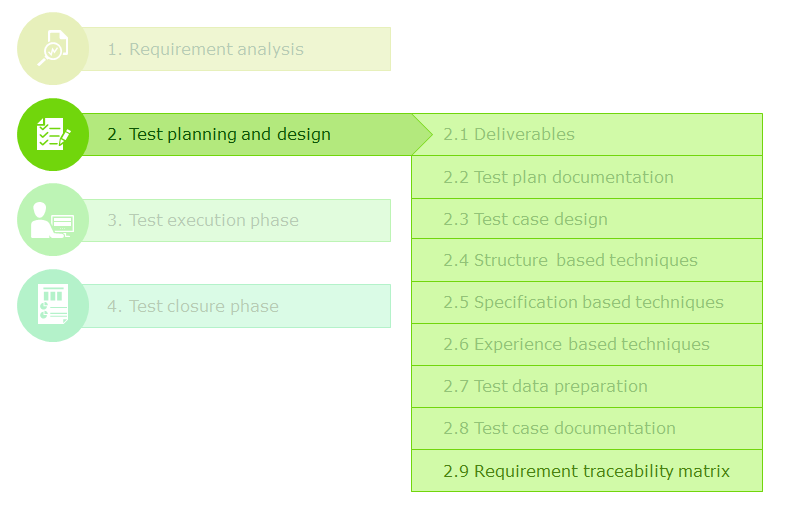
While entering data, following business rules should be followed.



**Assumptions to write test cases**

* Requirements naming  convention **R\_<Module Name>\_<Sequence Number>**
* Test case naming  convention **TC\_<Module Name>\_<Sequence Number>**
* Multiple messages are displayed for multiple scenarios occurring together.
* The user has successfully opened the ‘Oriental Business Hotels’ web site using the URL <www. Oriental Business Hotels.com > and is present on the Oriental Business Hotels ‘Add a Booking’ page.
* For mandatory field violation, error message displayed is: "Please enter the <<field name>>”.
* For invalid data, error message displayed is: “The entered <<field name>> is invalid”.

**2.9 Requirements Traceability Matrix**



**Requirements Traceability Matrix (RTM)** is a document, usually in form of a table that is prepared after designing test cases, to correlate the relationship between test cases and requirement specifications.

It simplifies and saves effort involved in a lot of QC tasks like reviews and audits, during the course of the software testing life cycle.

**Significance of RTM in test planning and design phase**

* Helps to audit and guarantee that all requirement specifications are covered by the test cases. If any requirement is uncovered, then design additional test cases.
* Helps to perform test focus analysis that ascertains which requirements are tested exhaustively and which requirements are tested feebly based on the number of test cases and then make necessary modifications according to the significance of the requirement.
* In case there is a change/correction in one of the requirement specifications the RTM helps in ascertaining the test cases that have to be reviewed, and if required, modified.
* It helps in risk management by helping to ascertain
  + Which requirements are at a risk of being untested if the required resources for a particular test are unavailable.
  + Which requirements are very critical for the testing process based on number of test cases that they impact.

**Significance of RTM in test execution phase**

* If all requirements are not completely developed then RTM helps to decide if the test execution phase can be started depending on the number of test cases impacted.
* If a defect is found by a test case then RTM helps in
  + Identifying the requirements impacted by the defect
  + Ascertaining the severity of the defect based on number of test cases impacted
  + Deciding which untested test cases will be blocked due to the defect
  + Deciding which test cases need testing again to ensure that they are still working as expected, once the defect is fixed.

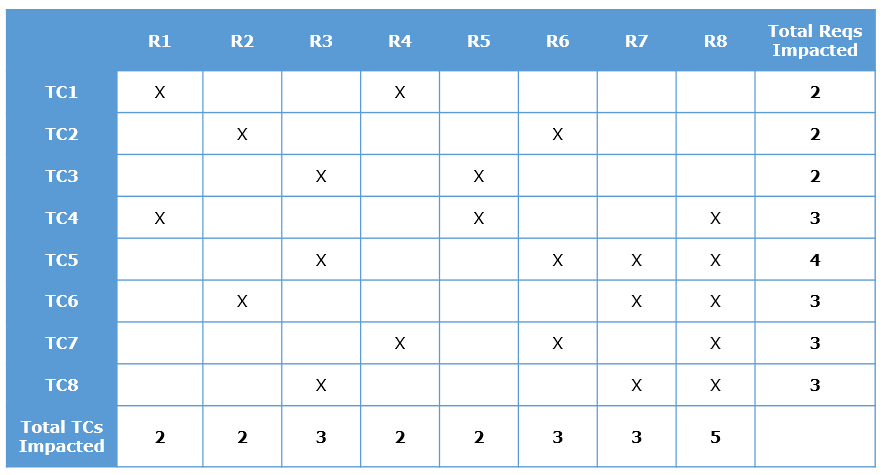
**Significance of RTM in test closure phase**

* Helps in identifying areas of focus for future tests based on
  + Defect distribution across the requirements
  + Test cases and strategies that need to be improved by identifying the test cases used to test requirements where there were defect slippages.

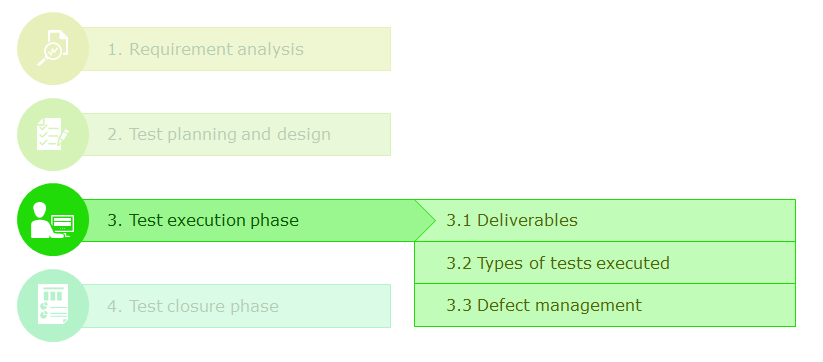
**Types of RTM**

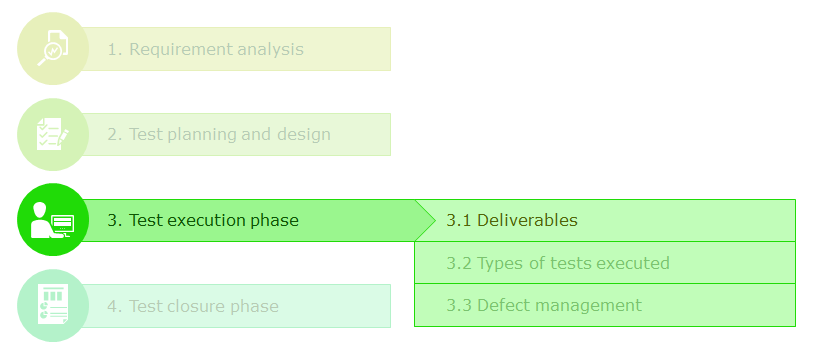
Based on the direction of traceability, RTMs can be classified as

1. **Forward Traceability RTM**: Enables tracing test cases from requirements. Usually implemented by documenting related test case IDs in the requirement document.
2. **Backward Traceability RTM**: Enables tracing requirements from test cases. Usually implemented by documenting requirement IDs in the test case document.
3. **Bidirectional RTM**: Requirements can be traced from test cases or vice versa. Usually implemented as a standalone document like the sample RTM below.



Test execution phase is the at the heart of STLC around which all other phases are defined. In this section, you will learn about the activities involved in the test execution phase.





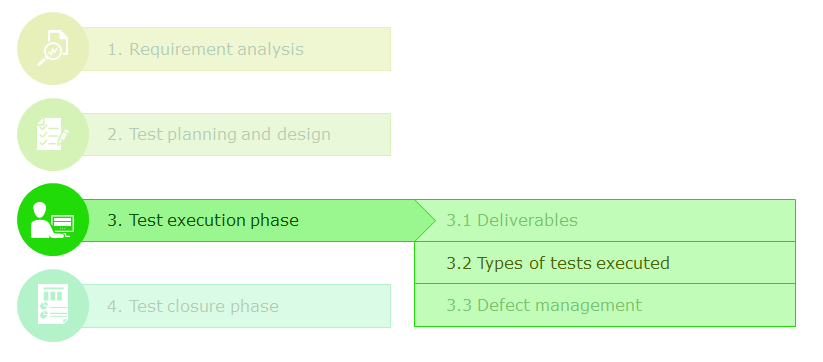
The testing team starts the system test execution phase after

* The test cases are completed and approved by the requirement owners.
* The development team has coded, unit-tested and deployed the software build in the test environment.

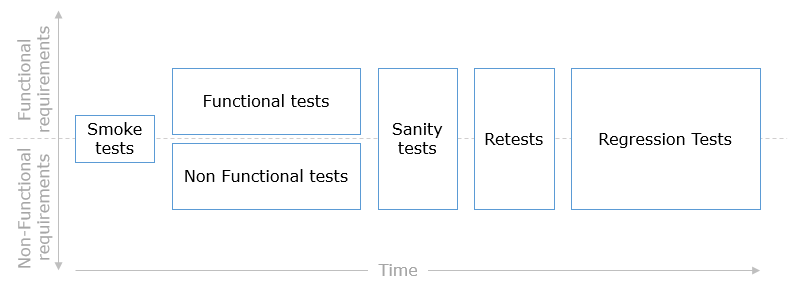
# Deliverables of test execution phase

* **Test execution results**  : Actual results of test cases documented in the test case document
* **Defects report**: List of defects found during test execution.

3.2 Types of Tests Executed



The testing team conducts an array of dynamic tests on the software product as described below.



**Smoke tests**

* Smoke tests are the first tests to be carried out during the test execution phase, immediately after the first code builds are delivered to the testing team.
* Their objective is to ascertain the software's test readiness - if the software is ready for the full blown testing that is about to follow.
* Smoke tests are performed on basic and critical components of the software. They are selected based on how quickly the tests can be carried out. For example
  + Whether the application is launching without any issues
  + Whether all the GUI components are available
  + Whether the response times of the application are within acceptable limits.
* Smoke tests can be informal (without documenting test cases and their execution results) or formal.

**Functional tests**

* Functional tests are the tests carried out to validate the functional requirements.
* In case of a newly created software (development project), these tests validate the entire software.
* In case of an existing software (maintenance project), the scope of these tests is restricted to the newly added, removed or modified components.

**Non-Functional Tests**

* Non functional tests are carried out to validate the non-functional requirements.
* They are normally carried out by a different group of testers (separate from regular black-box functional testers) as these tests may require specialized skills in terms of
  + Non-functional requirement analysis
  + Using white-box test techniques
  + Using specialized tools
* **Performance tests** are one of the most commonly conducted non-functional tests. They mostly consist of
  + **Load tests** - whether the system's response times are within acceptable limits when it is operated under regular load (concurrent number of users/transactions) for a specified duration.
  + **Stress tests** - whether the system's response times are within acceptable limits when it is operated under peak load for a specified duration.
  + **Spike tests** - whether the system's response times are within acceptable limits when the load suddenly increases or decreases by a huge difference.

**Sanity tests**

* Sanity tests are similar to smoke tests in terms of their objectives (validate test readiness) and their test selection (how quickly they can be executed).
* They differ from smoke tests in terms of their scope and time of execution.
* While smoke tests validate the entire software, sanity tests are limited to small components and defect fixes.
* While smoke tests are done on the initial large builds, sanity tests are carried out on smaller builds delivered in the later stages of test execution. If required, they can additionally test that no other critical functionality are broken by the defect fixes.
* Sanity tests are usually informal.

**Retests**

* Retests consist of re-running the failed test cases, once their corresponding defect fixes are delivered, to check if they are passing.

**Regression tests**

* While retests check whether failed test cases are passing after code changes, regression tests check whether passed test cases are still passing after code changes.
* The objective of regression tests is to make sure that none of the software components, which were working fine before the code change, are broken unintentionally.
* Informal and small regression tests are carried out as a part of sanity tests.
* Usually a formal regression test is carried out on the entire software once all the defects have been fixed, before the build is promoted to the next stage.
  + In case of development projects, formal regression tests are limited based on the impact analysis of the defects fixes.
  + In case of maintenance projects, formal regression tests are extensive and are carried out on all parts of the software which are not supposed to be impacted by the current software release.

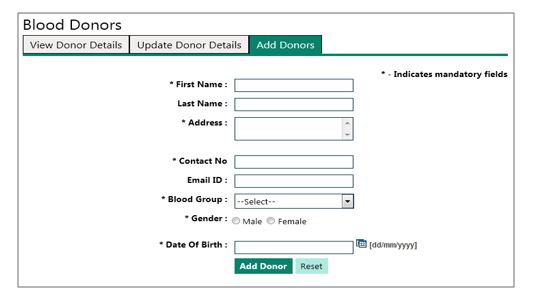
**Problem Statement:**

**Objective**

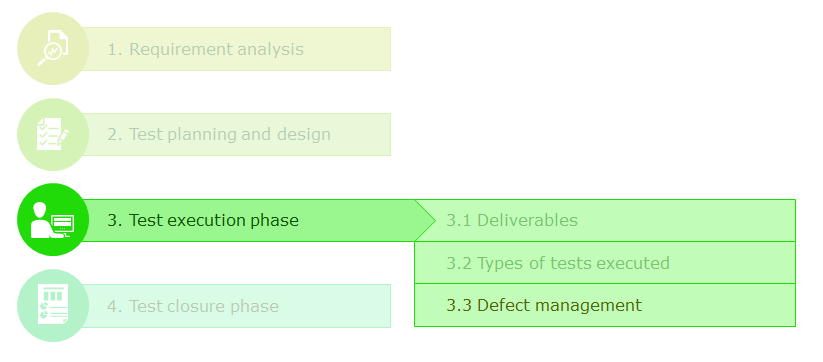
To understand the different types of tests that have to be performed on the software during test execution phase

**Problem statement**

Analyze the ‘Add Donor’ module of HMS (as in the below figure) and determine how and when the tester would use the various testing techniques like smoke, sanity, functional,regression.

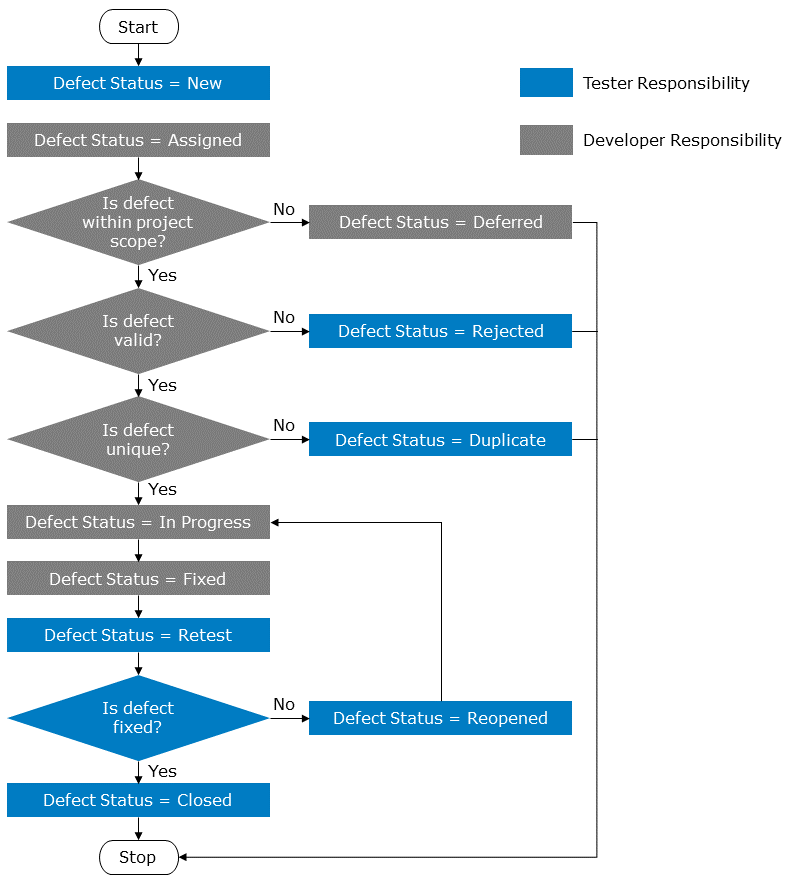


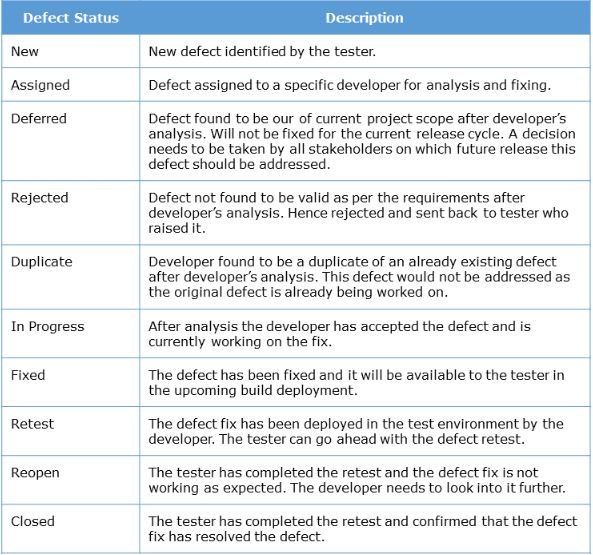
**3.3 Defect Management**



**Defect Management (DM)**is the process of logging, tracking and closing all the defects that are identified during the course of STLC.

Defect management processes and responsibilities are defined based on the life cycle of a defect.





Defect Logging

During test execution, whenever a test case fails due to a mismatch between the actual results and expected results, a defect MUST be logged.

**Defect logging** is the process of documenting all relevant information about the defect that needs to be passed on to the developer so that he/she can fix it.

The following field/column names describe the information that is captured about the defect in a systematic and organized way into a defect repository table or a defect management tool.

# ****Defect ID****

A numeric or alpha-numeric code that needs to be assigned to a defect so that the defect can be identified uniquely.

# ****Defect Title****

A meaningful one line title to state what the issue is. In situations where stating just the Defect ID does not help to recognize the defect due to it's cryptic nature, this defect title can be used. (Eg: Status calls, defect triage meetings, project check points etc.)

# ****Detected By/Opened By****

The name or identification of the person, usually the tester, who identified and opened the defect so that any stakeholder can contact that person in case of any additional information about the defect is needed.

# ****Opened Date****

The date on which the defect was identified. It can be used to check if the defects are addressed within the stipulated time as proposed in the test plan. A project risk may have to be raised if defect fixes are unavailable within the required time limits.

# ****Status****

Predefined value which states how much progress has been made with respect to getting the defect fixed. This information is constantly updated by the developers and testers whomever the defect is currently assigned to.

# ****Assigned To****

The name or identification of the person who is currently owning the defect based on its current status and his/her role in the project. This contact information can used by the project management to check the progress made on a particular defect on an ad hoc basis at any point in time.

# ****Priority****

This field is used to indicate how quickly the defect needs to be fixed. It is estimated based on the degree of risk the prolonged existence of the defect poses for

* Project schedule in terms of number of unexecuted tests it impacts/blocks.
* Amount of cycle time left in SDLC/STLC
* Brand value and quality perception of the product.

Priority is normally graded as

1. High
2. Medium
3. Low

# ****Severity****

This field is used to indicate the real world impact of the defect in terms of

* Number of product functionalities that would be rendered unusable
* Amount of financial loss it might result in

Severity is normally graded as

1. Critical/Showstopper
2. Major
3. Moderate
4. Minor
5. Cosmetic

# ****Defect Description****

This is the most descriptive field of a defect. It is filled with information pertaining to most, if not all the following details.

* Steps to reproduce the defect
* Test data which can be used to reproduce the defect
* Expected behavior and actual behavior
* Justification for the chosen priority and severity
* Any additional information that would help the developers in their analysis.

# ****Test Case ID****

This field is used for providing the test case ID of the test case which was used to identify the defect. It simplifies test management in circumstances where the initial test and defect retest might need to be executed by different testers. It also helps in post testing analysis in terms of analyzing the test case effectiveness, requirement traceablility, etc.

Defect Logging – Exercise

## Problem Statement:

# Objective

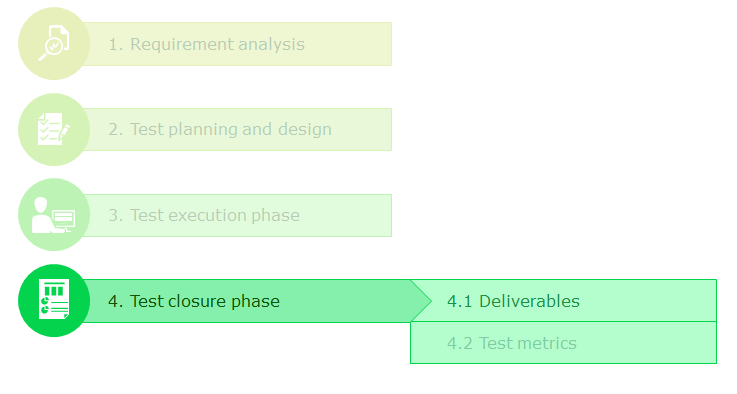
To understand the difference between the concepts of priority and severity of defects

# Problem statement

Assign the priority and severity values for the following defects. The values of priority can be assigned as HIGH, MEDIUM and LOW. The values of severity can be assigned as CRITICAL, MAJOR and MINOR

1. On providing the URL of the application in the address bar of the browser and hitting ENTER key, the application is not invoked. The error message "The page cannot be displayed" is coming up.
2. On entering all the customer details in the web page and clicking on 'Submit' button, the page is not getting refreshed and no action seems to be happening. This anomaly is happening only when using Mozilla Firefox however the functionality is working fine in Internet Explorer and Google Chrome.
3. The date of birth field in the customer information page is accepting values of future dates also. As per the requirements, its input values should have been restricted only to past dates.
4. In the home page of the web application, the image of the company logo is not loaded in the top right corner as per the requirements. Instead that space is empty.
5. In the application, the text fields sizes are different according to their maximum length allowed. These fields, though in a single column, do not start and end at the same margins. This is causing a misalignment of the fields and they are not appearing in a neat and organized way.

4.1 Deliverables



# Before closing the test project

* Evaluate the test exit criteria based on
  + Requirement coverage
  + Test completion
  + Status of defects

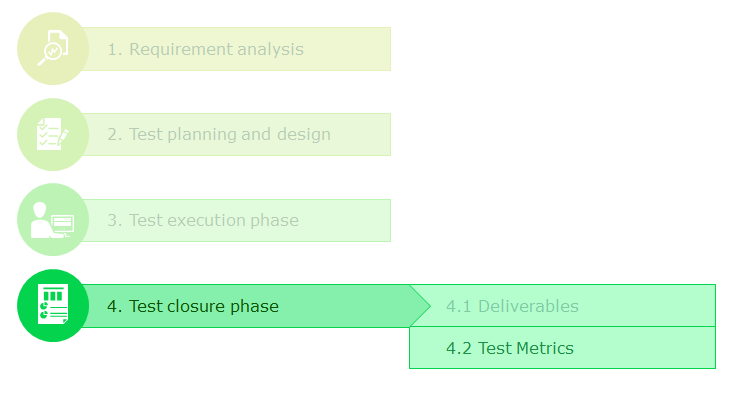
# After Closing the test project

* Archive all test artifacts and proof of testing for reuse/future reference.
* Analyze the defects' root cause and distribution.
* Analyze and document the metrics collected on test activities to identify areas to be improved.
* Document lessons learnt for enabling the success of future testing projects.

# Deliverables of test closure phase

* Approval/rejection for code promotion to the next phase based on exit criteria checklist.
* Metrics report
* Project lessons and future recommendations (based on situations)
* Case studies (optional)

Test Metrics



Though quality might seem a very subjective term to be used the truth about quality is that "if you cannot measure it then you cannot control it."

A metric is a quantitative measure of the degree to which a system, component of a system, or process possesses a given attribute.

To track and ensure the overall quality of the process of software testing itself, a number of metrics are collected and reviewed. These metrics can be broadly classified into two categories.

**Base Metrics**

These are metrics that can be directly derived (without any calculation). Base metrics are usually collected from the testers. Some of the most common base metrics collected are.

1. Total number of requirements
2. Total number of test cases generated
3. Total number of test cases executed
4. Total number of test cases passed/failed/blocked
5. Total number of defects
6. Total number of defects (based on severity/priority)
7. Total amount of effort spent on test case preparation
8. Total amount of effort spent on test execution

**Calculated Metrics**

These are metrics that are calculated from base metrics already collected. The calculations are normally performed by test leads and managers to monitor and report the health of the project. Some of most commonly collected metrics are

