SWE 4603

Software Testing and Quality Assurance

Lecture 1

Prepared By Maliha Noushin Raida, Lecturer, CSE
Islamic University of Technology

Classroom Code: **Idseke4**

Book: SOFTWARE TESTING Principles and Practices

By Naresh Chauhan

Week 1 On:
- Chapter 1: Introduction to Software Testing
- Chapter 2: Software Testing Terminology and Methodology

What is Software Testing

❖ Software Testing is a process where the software is evaluated to determine that it meets the user needs (validation) and that the process to build the software was followed correctly(verification).

❖ **"you built it right"** (verification)   process er majhe

❖ **"you built the right thing"** (validation)   process er sheshe

❖ Some standards and models to help with product quality include:
   ✔ IEEE Std 1012-2012 IEEE Standard for System and Software Verification and Validation
   ✔ Software Engineering Institute (SEI) Capability Maturity Model (CMM)

Definition

Testing is the process of executing a program with the intent of finding errors.   - Myers

Testing is a support function that helps developers look good by finding their mistakes before anyone else does.

- James Bach

Software testing is a process that detects important bugs with the objective of having better quality software.

Testing: Why Do We Perform

Testing fulfills two primary purposes:

❖ To demonstrate quality or proper behavior;
❖ To detect and fix problems

Finding and eventually fixing defects is the Number one reason for software testing. But the truth is, no Software is free from Defect.

Then Why so much effort to test and invest money in it?

Testing: Why It Is Important

This can have a huge impact on a company, including:

❖ **Financial Loss:** Losing business is bad enough but in extreme cases there are financial penalties for non-compliance to regulations

❖ **Increased costs:** These tend to be the result of having to implement a manual workaround but can include staff not being able to work due to a fault or failure and the cost of fixing the issues after go-live

❖ **Reputational loss:** If the company is unable to provide service to their customers due to software problems then the customers will probably take their business elsewhere

Testing: Why It Is Important

❖ Insufficient testing and the resulting errors and failures of the product, can have significant impacts on the business.

❖ Inadequate software testing costs the US alone between $22 and $59 billion annually and that better approaches could cut this amount in half.

❖ The later in the development cycle an error is found, the more expensive it is to fix.

Cost of Late Testing

❖ Sometimes the defect is not fixed at all because of the costs involved.

❖ when a defect is found during the user requirements analysis, all it takes to fix it is to update the requirements document.

❖ However, if that same defect is missed until the whole product is developed, fixing it may require a change in the design, a rewrite of significant amount of code, customer re-approvals, and re-testing of the product once the fixes are complete.

❖ The more complex and more expensive is the product, the greater is the cost of finding defects late

Cost of Late Testing

❖ IBM mainframe z/OS system defect that was found by a customer in the field was estimated to cost at minimum 10K per defect.

❖ Researchers found that if finding an error in requirements phase costs 1 unit then it increases so significantly that the same defect if undetected until operations phase can cost to fix anywhere from 29 units to more than 1,500 units.

❖ So if a unit costs $1000 then at requirements phase a defect would cost a $1,000 to fix while in the operations it can cost from $29,000 to $150,000 to fix.

Deadly Software Failure

Radiotherapy system Therac-25

Six accidental releases of radiation (1985-1987) killing three patients. The actual code error was a counter in a routine which would often overflow. If the machine's operator inputted something when this happened, the software safeguard would fail. As a result the patient would get 100 times greater dosage of radiation then he should.
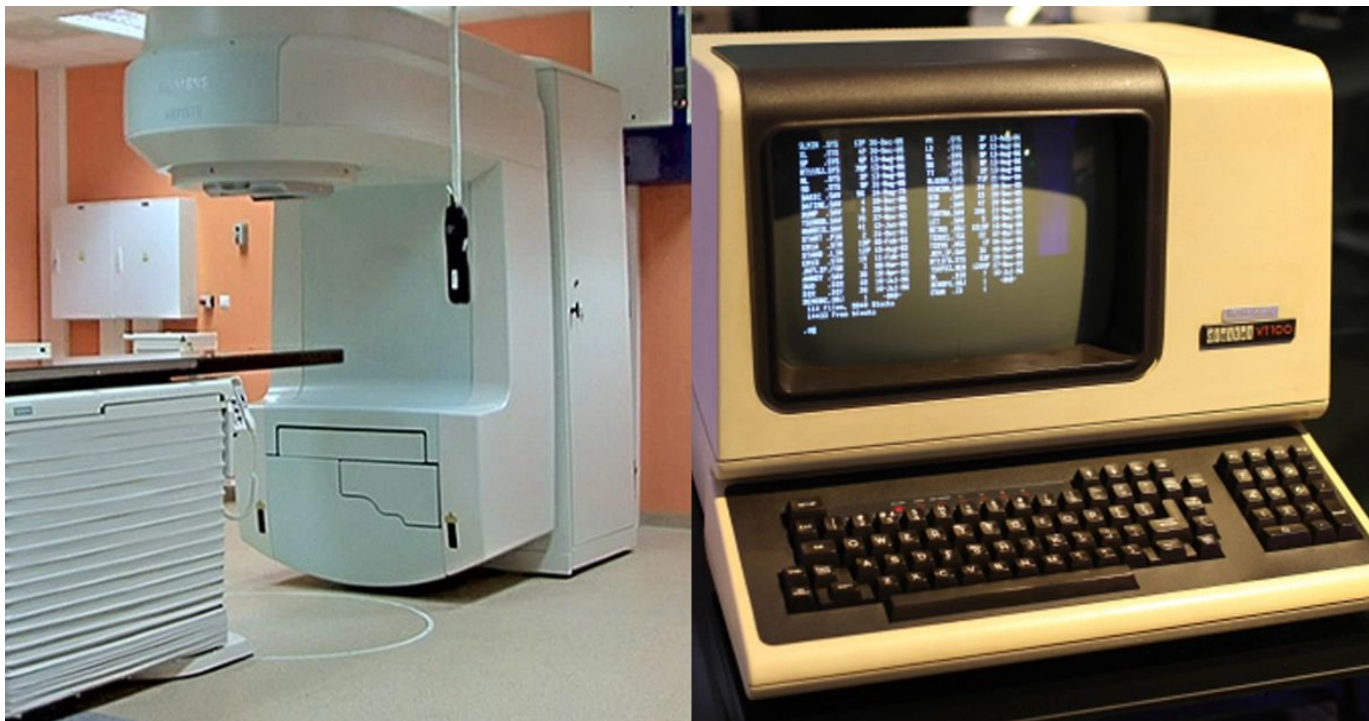
Fig: Therac-25

Software Testing - Myths and Facts

Myths

> Testing is a single phase in SDLC

Facts

> In reality, testing starts as soon as we get the requirement specifications for the software. And the testing work continues throughout the SDLC, even post-implementation of the software.

Myths

> Testing is easy.

Facts

> It is not easy, as they have to plan and develop the test cases manually and it requires a thorough understanding of the project being developed with its overall design. Overall, testers have to shoulder a lot of responsibility which sometimes make their job even harder than that of a developer.

Software Testing - Myths and Facts

Myths

Software development is worth more than testing.

Facts

Testing is a complete process like development, so the testing team enjoys equal status and importance as the developers.

Myths

Complete testing is possible.

Facts

There are many things which cannot be tested completely, as it may take years to do so.

Myths

Testing starts after program development.

Facts

The tester performs testing at the end of every phase of SDLC in the form of verification and plans for the validation testing

Software Testing - Myths and Facts

Myths

The purpose of testing is to check the functionality of the software.

Facts

Quality does not always imply checking only the functionalities of all the modules. There are various things related to quality of the software, for which test cases must be executed

Myths

Anyone can be a tester.

Facts

WE WILL SEE ABOUT THAT

Goals Of Software Testing

TESTING

Immediate Goals
- Bug discovery
- Bug prevention

Post-implementation Goals
- Reduced maintenance cost
- Improved testing process

Long-term Goals
- Reliability
- Risk management
- Quality
- Customer satisfaction

Testing Terminology

Failure

It Is the inability of a system or component to perform a required function according to its specification.

Fault/Defect/Bug

It is a fault due to an **error** in code whether due to incorrect design, logic, or implementation where the software does not behave as expected.

Error

Whenever a development team member makes a mistake in any phase, errors are produced. Example: typographical error, a misleading of a specification, a misunderstanding of what a subroutine does, and so on.

Error → Bug → Failures

16
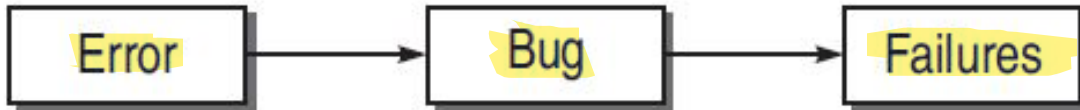
Why Do Bugs Occur?

Faulty definition of requirements

- Erroneous requirement definitions
- Absence of important requirements
- Incomplete requirements
- Unnecessary requirements included

Client-developer communication failures

- Misunderstanding of client requirements presented in writing, orally, etc.
- Misunderstanding of client responses to design problems

Coding errors

- Errors in interpreting the design document, errors related to incorrect use of programming language constructs, etc.

Why Do Bugs Occur?

Deliberate deviations from software requirements

- Reuse of existing software components from previous projects without complete analysis

- Functionality omitted due to budge or time constraints

- "Improvements" to software that are not in requirements

Logical errors

- Errors in interpreting the requirements into a design (e.g. errors in definitions of boundary conditions, algorithms, reactions to illegal operations,...)

Shortcoming in testing

- Incomplete test plan
- Failure to report all errors/faults resulting from testing
- Incorrect reporting of errors/faults
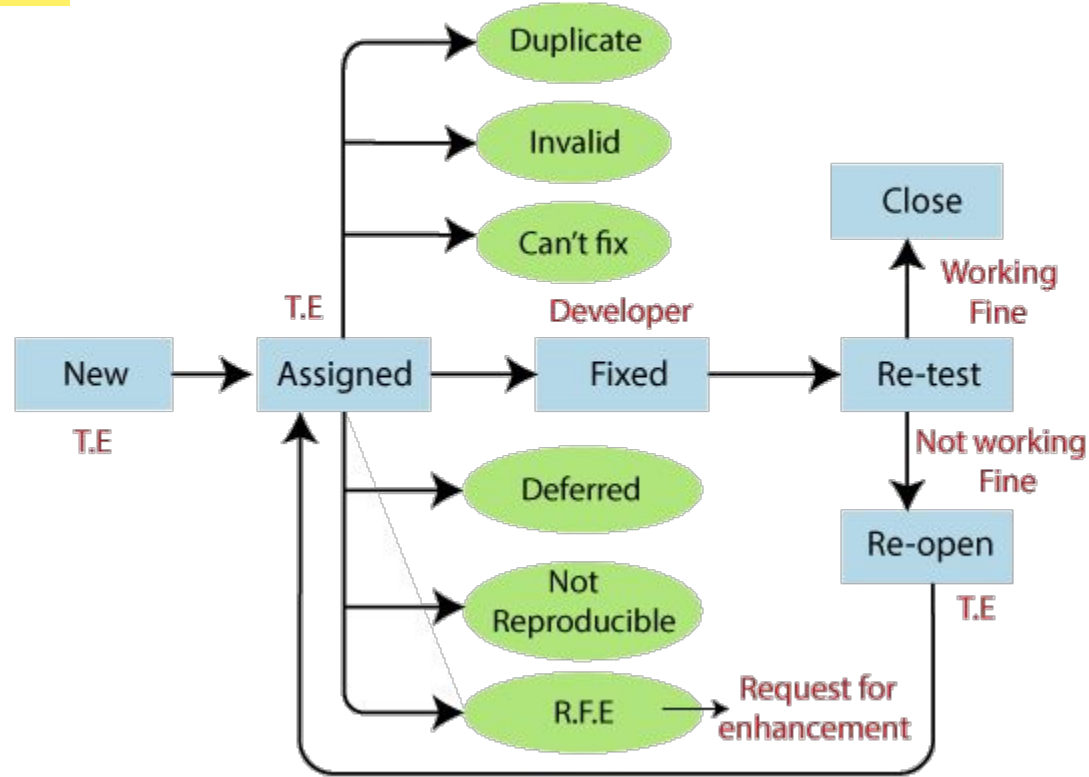- Incomplete correction of detected errors

Bug Life Cycle
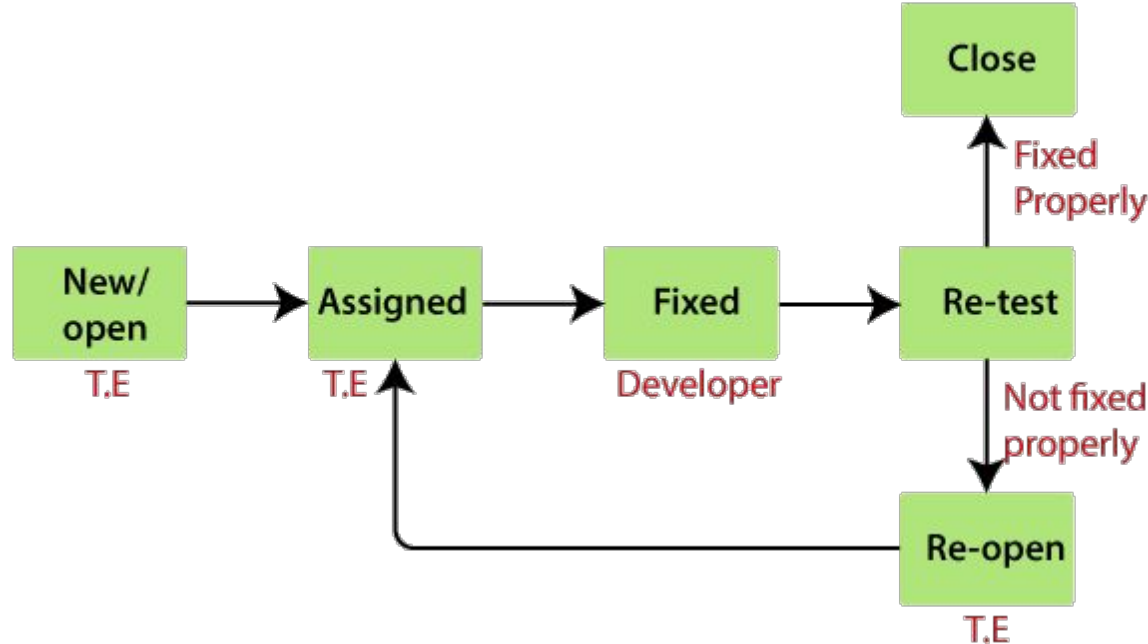
Following are the **different status** of the bug:
- Invalid/rejected
- Duplicate
- Postpone/deferred
- Can't fix
- Not reproducible
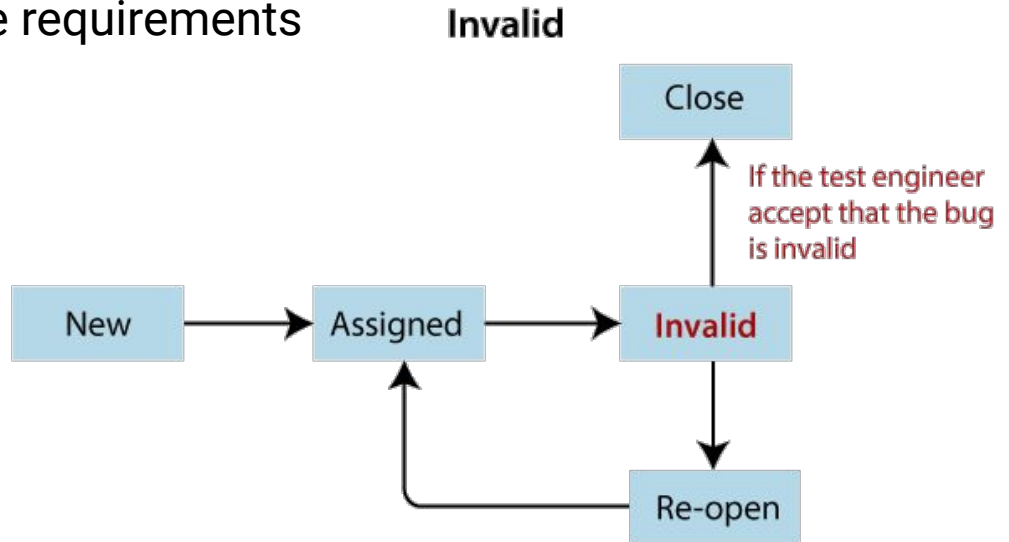- RFE (Request for Enhancement)

Bug Life Cycle

Invalid Bug

Any bug which is not accepted by the developer is known as an invalid bug.

Reasons for an invalid status of the bug :

- Test Engineer misunderstood the requirements
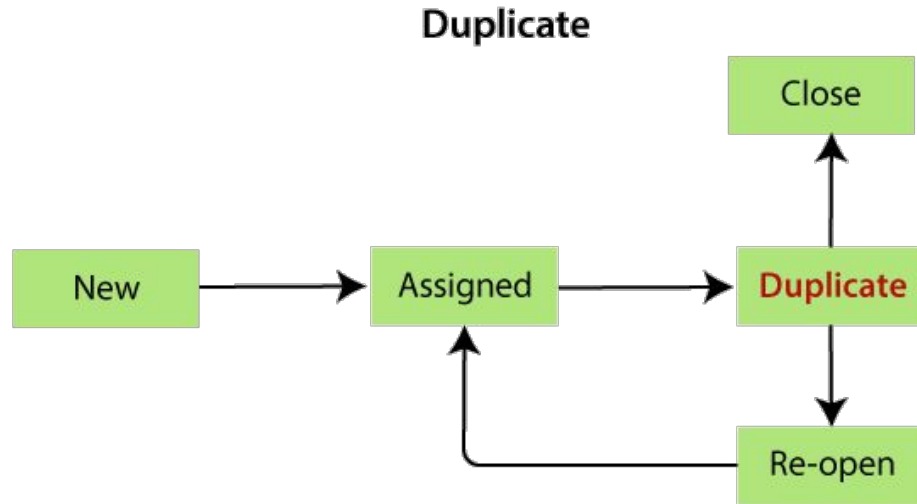
- Developer misunderstood the requirements

Invalid

Close

If the test engineer accept that the bug is invalid

New → Assigned → Invalid

Invalid → Re-open → Assigned

Duplicate Bug

Reasons for an duplicate status of the bug :

- Common features

- Dependent Modules

22

Duplicate Bug

To avoid the duplicate bug:

- ❖ If the Developer got the duplicate bug, then he/she will go to the bug repository and search for the bug and also check whether the bug exist or not.

- ❖ If the same bug exist, then no need to log the same bug in the report again.

- ❖ If the bug does not exist, then log a bug and store in the bug repository, and send to Developers and Test Engineers adding them in [CC].

Not Reproducible

These are the bug where the developer is not able to find it, after going through the navigation step given by the test engineer in the bug report.

Reasons for the not reproducible status of the bug
- **Incomplete bug report**
  The Test engineer did not mention the complete navigation steps in the report.
- **Environment mismatch**
  Environment mismatch can be described in two ways:
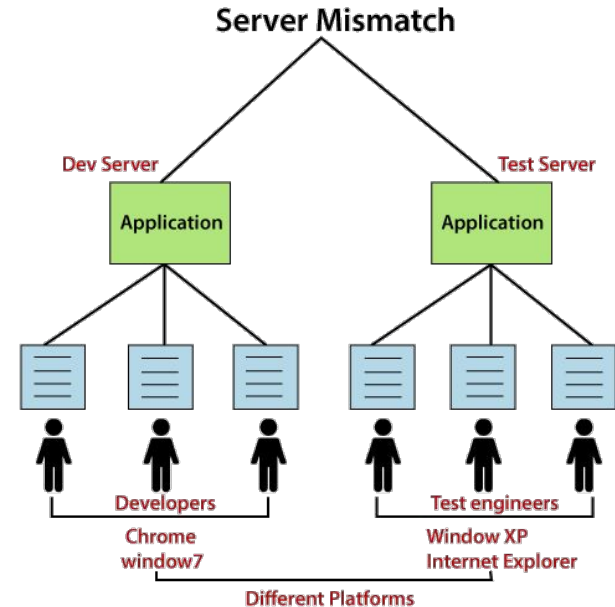   - Server mismatch
    - Platform mismatch

Not Reproducible

Server mismatch

Test Engineer is using a different server (**Test Server**), and the Developer is using the different server (**Development Server**) for reproducing the bug

Platform mismatch:
Test engineer using the different Platform (**window 7 and Google Chrome browser**), and the Developer using the different Platform (**window XP and internet explorer**) as well.

Server Mismatch

Dev Server   Test Server

Application   Application

Developers   Test engineers

Chrome window7   Window XP Internet Explorer

Different Platforms

Not Reproducible

Data mismatch
Different Values used by test engineer while testing & Developer uses different values.
For example:
The requirement is given for admin and user.

Test engineer(user) using the below requirements:	**the Developer (admin)** using the below requirements:
Username → abc Password → 123	Username → aaa Password → 111

Not Reproducible

Build mismatch
The test engineer will find the bug in one Build, and the Developer is reproducing the same bug in another build. The bug may be automatically fixed while fixing another bug.

Inconsistent bug
The Bug is found at some time, and sometime it won't happen.

Solution for inconsistent bug
As soon as we find the bug, first, **take the screenshot**, then developer will **re-confirm** the bug and fix it if exists.

Can't Fix

When Developer accepting the bug and also able to reproduce, but can't do the necessary code changes due to some constraints.

Reasons for the can't fix status of the bug

- **No technology support:** The programming language we used itself not having the capability to solve the problem.

- **The Bug is in the core of code (framework):** If the bug is **minor** (not important and does not affect the application), the development lead says it can be fixed in the next release.
  Or , if the bug is **critical** (regularly used and important for the business) and development lead cannot reject the bug.

- **The cost of fixing a bug is more than keeping it.**

28
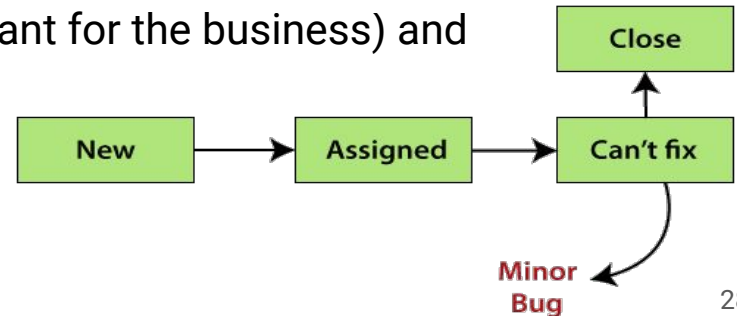
Deferred/Postponed

The deferred/postpone is a status in which the bugs are postponed to the future release due to time constraints.
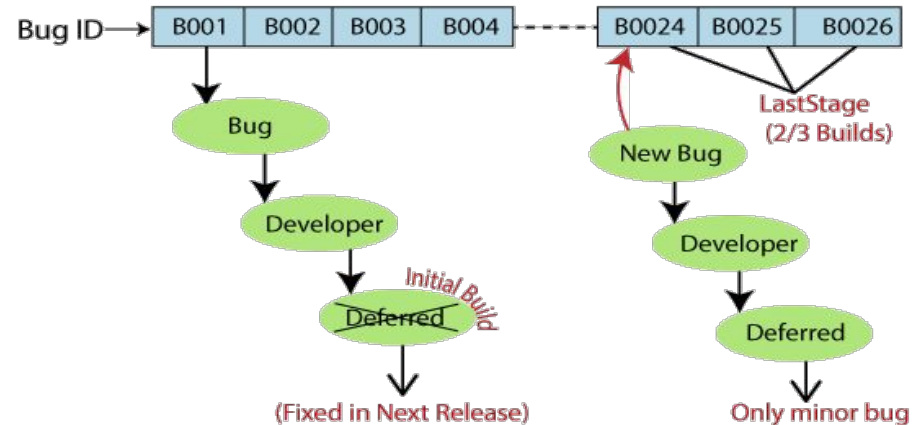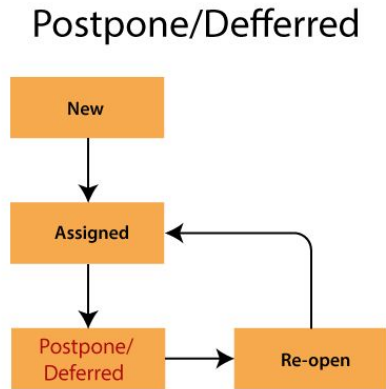
- The **Bug ID-B001** bug is found at the initial build, but it will not be fixed in the same build, it will postpone, and **fixed in the next release**.
- And **Bug ID- B0024, B0025, and B0026** are those bugs, which are found in the last stage of the build, **and they will be fixed because these bugs are the minor bugs**.

RFE(Request For Enhancement)

These are the suggestions given by the test engineer towards the enhancement of the application in the form of a bug report. The RFE stands for Request for Enhancement.
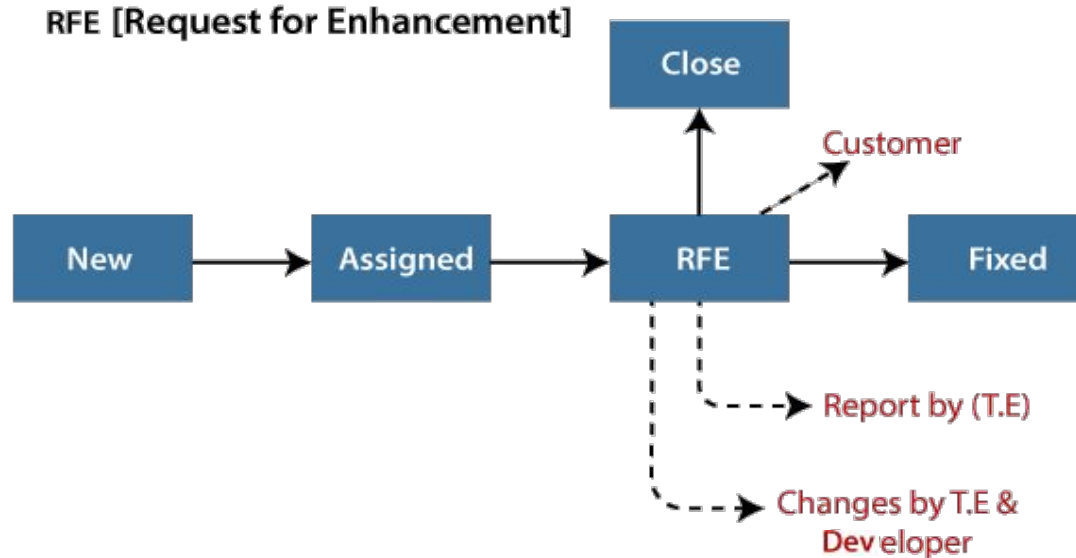
RFE(Request For Enhancement)

As we can see in the below image that the test engineer thinks that the look and feel of the application or software are not good because the test engineer is testing the application as an end-user, and he/she will change the status as RFE.
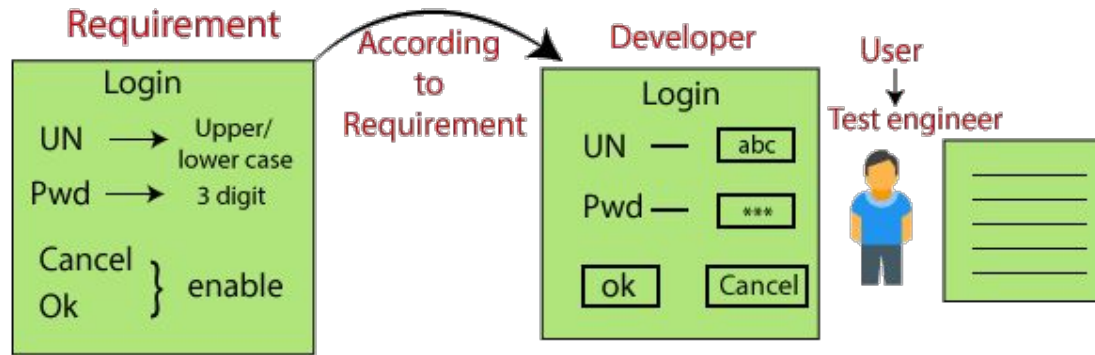
And if the customer says Yes, then the status should be Fix.
Or
If the customer says no, then the status should be Close.

Bug Severity

The impact of the bug on the application is known as severity.

It can be a blocker, critical, major, and minor for the bug.

Blocker: if the severity of a bug is a blocker, which means we cannot proceed to the next module, and unnecessarily test engineer sits ideal.

Critical: if it is critical, that means the main functionality is not working, and the test engineer cannot continue testing.

Major: if it is major, which means that the supporting components and modules are not working fine, but test engineer can continue the testing.

Minor: if the severity of a bug is major, which means that all the U.I problems are not working fine, but testing can be processed without interruption.

Bug Priority

Priority is important for fixing the bug or which bug to be fixed first or how soon the bug should be fixed.
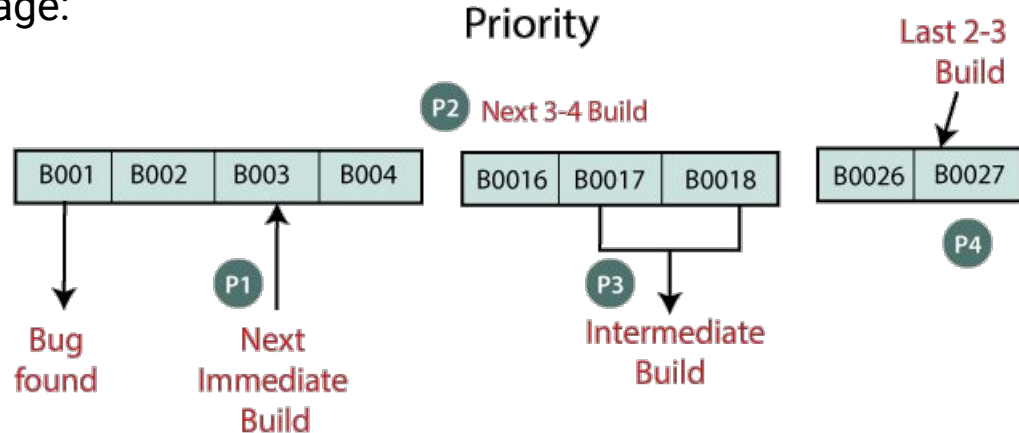
It can be urgent, high, medium, and low.

High: it is a major impact on the customer application, and it has to be fixed first.

Medium: In this, the problem should be fixed before the release of the current version in development.

Low: The flow should be fixed if there is time, but it can be deferred with the next release.

Example Of Priority and Severity

- When the bug is just found, it will be fixed in the next immediate build, and give the Priority as P1 or urgent.

- If the Priority of the bug is P2 or high, it will be fixed in the next 3-4 builds.

- When the Priority of the bug is P3/medium, then it will be fixed in the intermediate build of the application.

- And at last, if the Priority is P4/low, it will be fixed in the last 2-3 build of the software as we can see in the below image:

Priority

Last 2-3 Build

P2 Next 3-4 Build

B001	B002	B003	B004

B0016	B0017	B0018

B0026	B0027

P1

P3

P4

Bug found

Next Immediate Build

Intermediate Build

Example Of Priority and Severity

If we take the case of a mailing application, then the severity and Priority could depend on the application like as we can see in the below image:

Severity	Requirement	Priority
Critical ←	Login →	[P1]
Critical ←	Compose →	[P1]
Critical ←	Inbox →	[P1]
Major ←	Send Item →	[P2]
Major ←	Trash →	[P3]
Minor ←	Help →	[P3]
Minor ←	Logout →	[P4]

Writing Defects/Bugs

What makes a good defect report?

Often defect reports do not have enough information or they are inaccurate and so they are either rejected or when reviewed later, it is impossible to say what the actual problem was.

Parameters of Reporting a Bug/Defect

The Following details should be part of a Bug:

- Date of issue, author, approvals and status.

- Severity and priority of the incident.

- The associated test case that revealed the problem

- Expected and actual results.

- Description of the incident with steps to Reproduce

- Status of the incident

- Conclusions, recommendations and approvals.

Characteristics of a Good Bug Report

- **Objective** – criticizing someone else's work can be difficult. Care should be taken that defects are objective, non-judgmental and unemotional. e.g. don't say "your program crashed" say "the program crashed" and don't use words like "stupid" or "broken".
- **Specific** – one report should be logged per defect and only one defect per report.
- **Concise** – each defect report should be simple and to-the-point. Defects should be reviewed and edited after being written to reduce unnecessary complexity.
- **Persuasive** –the pinnacle of good defect reporting is the ability to present them in a way which makes developers want to fix them

Characteristics of a Good Bug Report

- **Reproducible** – the single biggest reason for developers rejecting defects is because they can't reproduce them. As a minimum, a defect report must contain enough information to allow anyone to easily reproduce the issue if it can be reproduced at will. Timing issues may be hard to reproduce but in that case there should be other information that can help in determining where in the code the error may be

- **Explicit** – defect reports should state information clearly or they should refer to a specific source where the information can be found. e.g. "click the button to continue" implies the reader knows which button to click, whereas "click the 'Next' button" explicitly states what they should do.

Sample Bug Report

Bug Report Template	
Bug ID	
Module	
Requirements	
Test Case Name	
Release	
Version	
Status	
Reporter	
Date	
Assign To	
CC	
Severity	
priority	
Server	
Platform	
Build No.	
Test Data	
Attachment	
Brief Description	
Navigation Steps	
Observation	
Expected Result	
Actual Result	
Additional Comments	

Bug Management

- Defects need to be handled in a methodical and systematic fashion

- There's no point in finding a defect if it's not going to be fixed

- There's no point getting it fixed if you don't know it has been fixed and there's no point in releasing software if you don't know which defects have been fixed and which remains.

What to do to manage?

The answer is to have a Bug tracking system

The simplest can be a database or a spreadsheet. A better alternative is a dedicated system, which enforce the rules and process of defect handling and makes reporting easier. Some of these systems are costly but there are many freely available variants.

Test Case

Test Plan which spells out all the details of the testing to be done, schedules and milestones, responsibilities, testing systems and installation and setup, and the list of test cases to be executed. A **test case** is a set of actions executed to verify a particular feature or functionality of your software application.

- Test cases can be manual where a tester follows conditions and steps by hand or automated where a test is written as a program to run against the system.

- Tests are usually first defined in a document and often have to be approved by developers and/or testers.

- Automated tests may have all the documentation in the code implementing the test case and in the comments.

Fields of a Test Case

- **Unique Test Case name and/or ID:** Each test case needs an identifier. Some tests will have both an id and a meaningful name which is derived from the requirement or the functionality being tested.
- **Test Scenario and test summary or description:** This description should specify what behavior/functionality will be tested by this test case.
- **Pre-requisites or setup:** What needs to be setup first, such as previous input or commands to the system to execute this test case.
- **Test data or inputs:** Data to be used for this test case.
- **Execution Steps:** The steps that have to be done against the system to test the behavior for this test case.
- **Expected behavior/Result:** what is expected to happen after execution
- **Assumptions:** Any assumptions that are made about the system or the test case. For example, data dependency or other test case dependency.
- **Actual results:** what actually happened when the test case was run.
- **Status:** What is the current status of the test case such as passed, failed, or not tested

Characteristics of Good test case

There are a number of guidelines or best practices in writing efficient test cases.

- **Only test one thing in a test case**:
  Your test case should not be complicated and it should only test a single condition or value. It should be as "atomic" as possible and it should not overlap other test cases either

- **Test case should have an exact and accurate purpose**
  There should be no confusion what the test is supposed to be checking and what the expected behavior and result should be. The test case should be accurate on what it tests and it should test what it is intended to test.

- **Test case should be written in a clear and easy to understand language**
  This applies to both the test definition in the documentation and in the actual code. Just like the test definition should be clear, the code that implements it should be straightforward and clear. Any developer or tester should be able to understand exactly what the test case is trying to do by reading it once.

Characteristics of Good test case

- **Test case should be relatively small**
  If your test case is following the rule for only testing a single thing, it should not be large. If you find yourself writing a long test case, then chances are you are trying to test too much. In that case you need to break it down into multiple test cases.
- **Test case should be independent**
  You should be able to execute the test cases in any order and independently of other test cases. This makes the test case simpler as it is self-sufficient and there is no reason to track or worry about any other test cases.
- **Test case should not have unnecessary steps or words**
  The test case should be precise and economical and only have what it needs to have to describe what it is testing and how it should be tested. It should not be cluttered with any unnecessary and confusing verbiage.

Characteristics of Good test case

- **Test case should be traceable to requirements or design**

  The test case needs to test some behavior or characteristic that the code is supposed to have. So that means that there is an explicit requirement from the user or product owner, explicit organization requirement (e.g. serviceability for every product), or implicit requirement derived from some explicit requirement that the test case can be traced to.
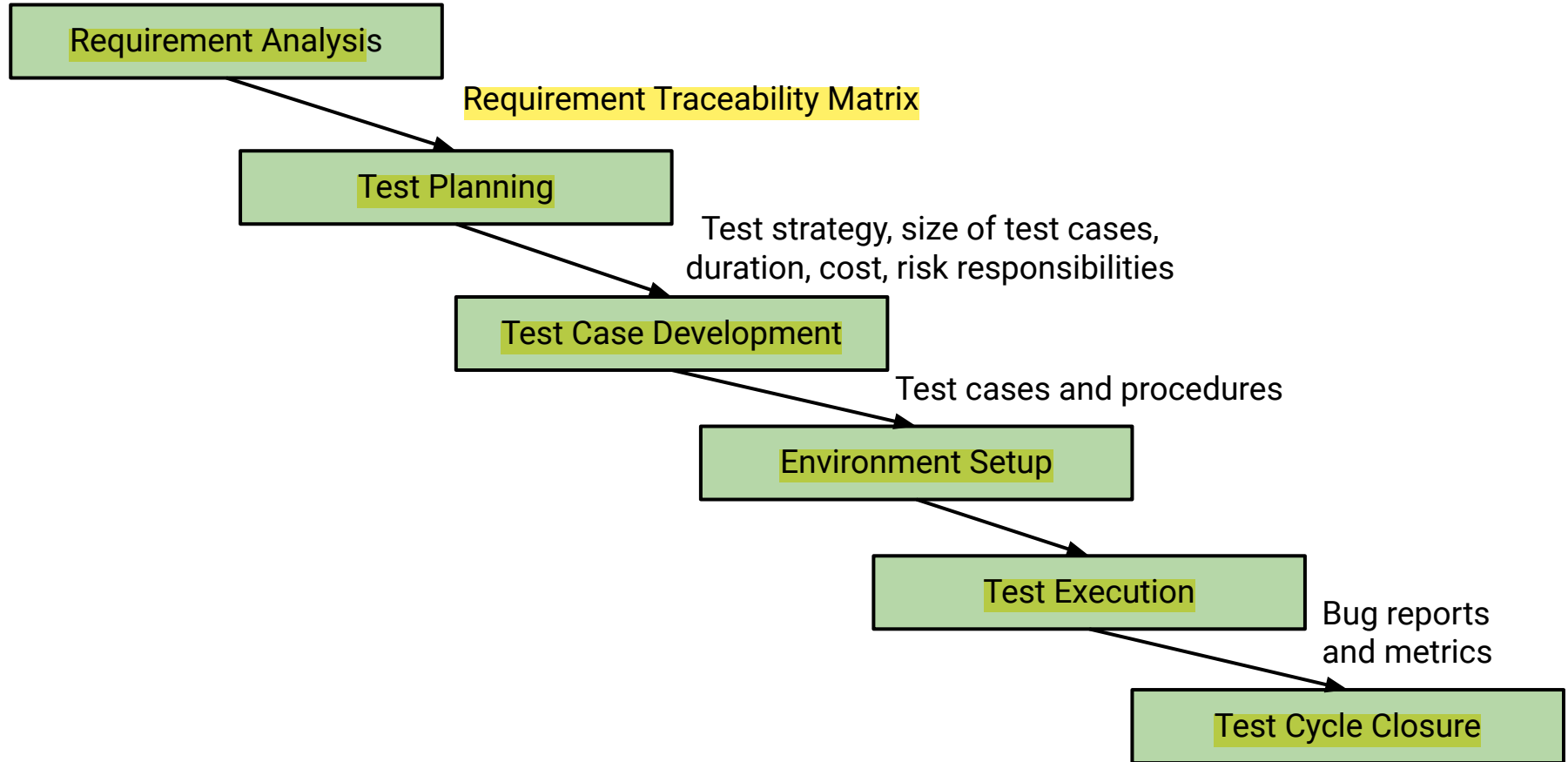
- **Test case should be repeatable**

  The test cases should be able to be rerun anytime and in any order. This allows for regression testing, reruns for fixes, and continuous integration where tests are run every time changed code is integrated into the product.

- **Test case should use consistent terminology and identification of functionality**

  When naming or identifying a feature, functionality, or widget, there should be the same and consistent terminology within the test case and across test cases. So for example, if test cases are describing the tests for login page for a college class, they should not have "user", "person", and "student" to refer to the same identity.

46

Software Testing Life Cycle

Requirement Analysis

Requirement Traceability Matrix

Test Planning

Test strategy, size of test cases,
duration, cost, risk responsibilities

Test Case Development

Test cases and procedures

Environment Setup

Test Execution

Bug reports
and metrics

Test Cycle Closure

Entry and Exit Criteria in STLC

Entry Criteria: Entry Criteria gives the prerequisite items that must be completed before testing can begin.

Exit Criteria: Exit Criteria defines the items that must be completed before testing can be concluded.

In an Ideal world, you will not enter the next stage until the exit criteria for the previous stage is met. But practically this is not always possible.

Requirement Analysis

Test team studies the requirements from a testing point of view to identify testable requirements and the QA team may interact with various stakeholders to understand requirements in detail.

Entry Criteria
- Requirements Specification.
- Application architectural

REQUIREMENT ANALYSIS
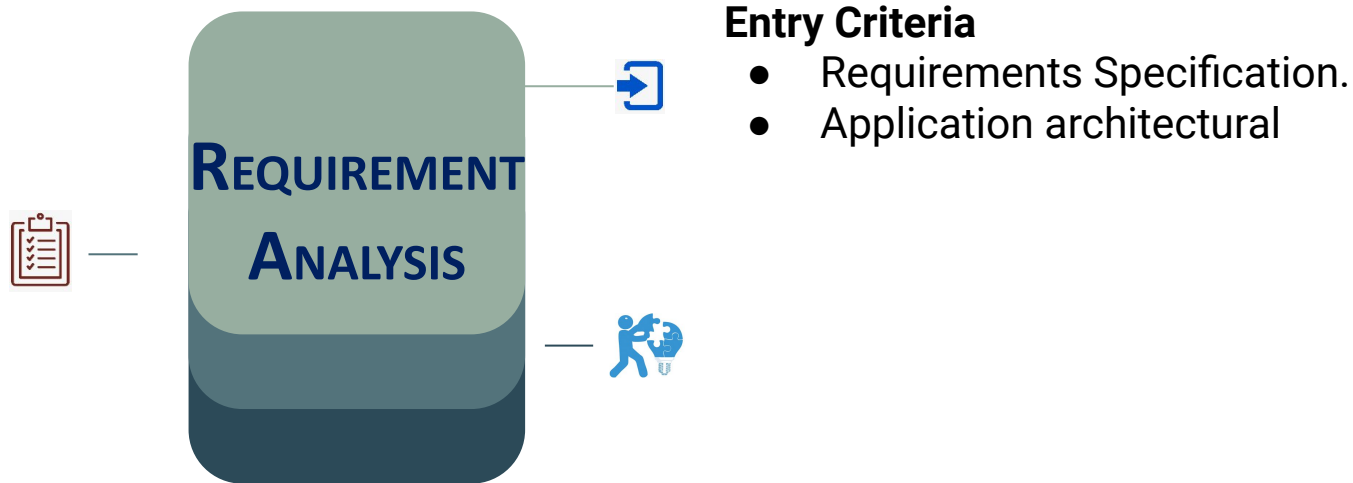
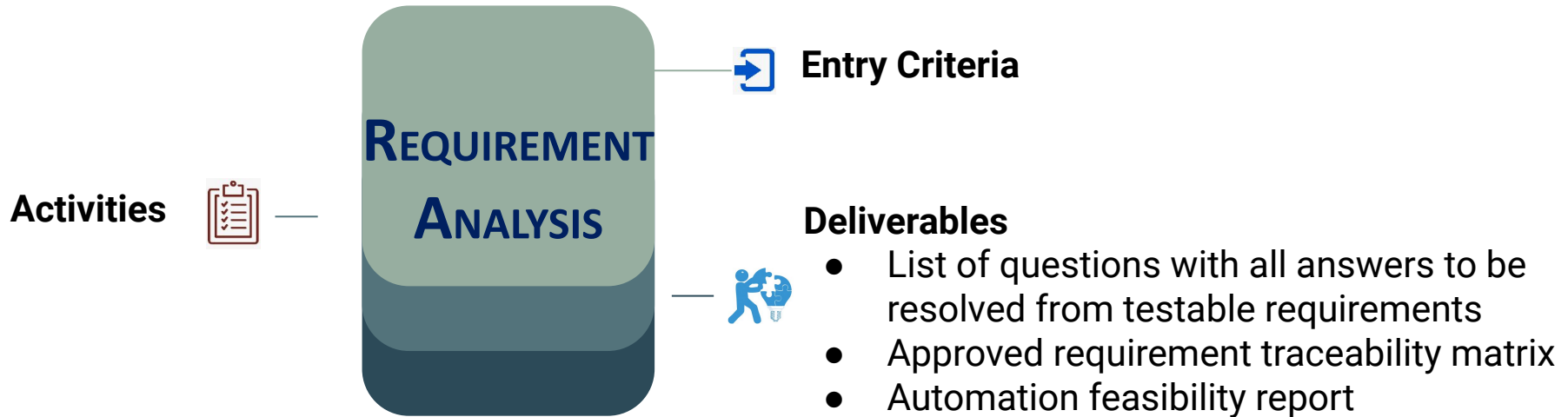# Requirement Analysis

Activities
- Identify types of tests to be performed.
- Gather details about testing priorities and focus.
- Prepare Requirement Traceability Matrix (RTM).
- Identify test environment details where testing is supposed to be carried out.
- Automation feasibility analysis (if required).

REQUIREMENT ANALYSIS

→ **Entry Criteria**

Requirement Analysis

Activities

REQUIREMENT ANALYSIS

Entry Criteria

Deliverables
- List of questions with all answers to be resolved from testable requirements
- Approved requirement traceability matrix
- Automation feasibility report

Testing Planning

A Senior QA manager determines the test plan strategy along with efforts and cost estimates for the project. Moreover, the resources, test environment, test limitations and the testing schedule are also determined.

TEST **P**LANNING

Entry Criteria
- Updated requirements document.
- Test feasibility reports
- Automation feasibility report.
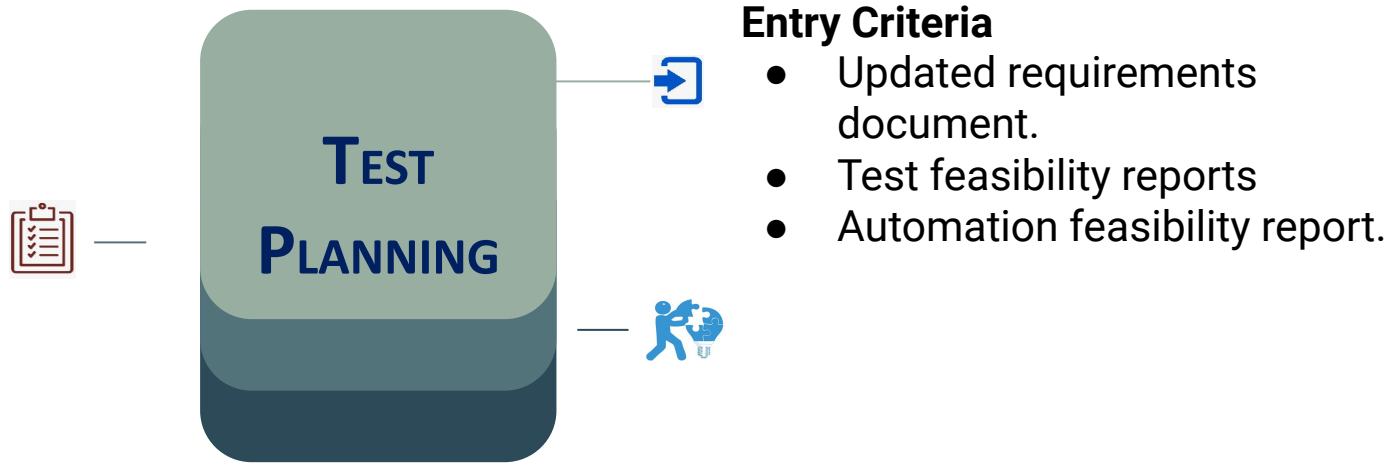
Testing Planning

Activities

- Preparation of test plan/strategy document for various types of testing
- Test tool selection
- Test effort estimation
- Resource planning and determining roles and responsibilities.
- Training requirement.

TEST PLANNING

→ **Entry Criteria**

Testing Planning

Activities

TEST PLANNING

Entry Criteria

Deliverables
- Test plan /strategy document.
- Effort estimation document.

Test Case Development

The Test data is identified then created and reviewed and then reworked based on the preconditions. <mark>Then the QA team starts the development process of test cases for individual units.</mark>

TEST CASE DEVELOPMENT

Entry Criteria
- Test plan ready and signed
- Requirement documents
- RTM
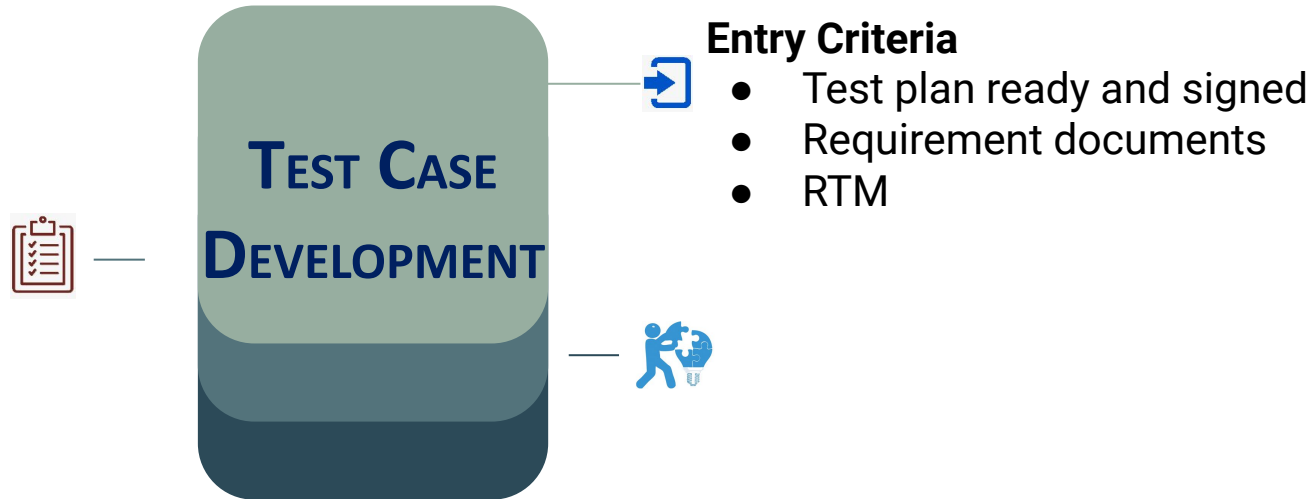
Test Case Development

Activities
- Create test cases, automation scripts (if applicable)
- Review and baseline test cases and scripts
- Create test data (If Test Environment is available)

TEST CASE DEVELOPMENT

→ **Entry Criteria**

Test Case Development

Activities

Test Case Development

Entry Criteria

Deliverables
- Test Scripts
- Test Data
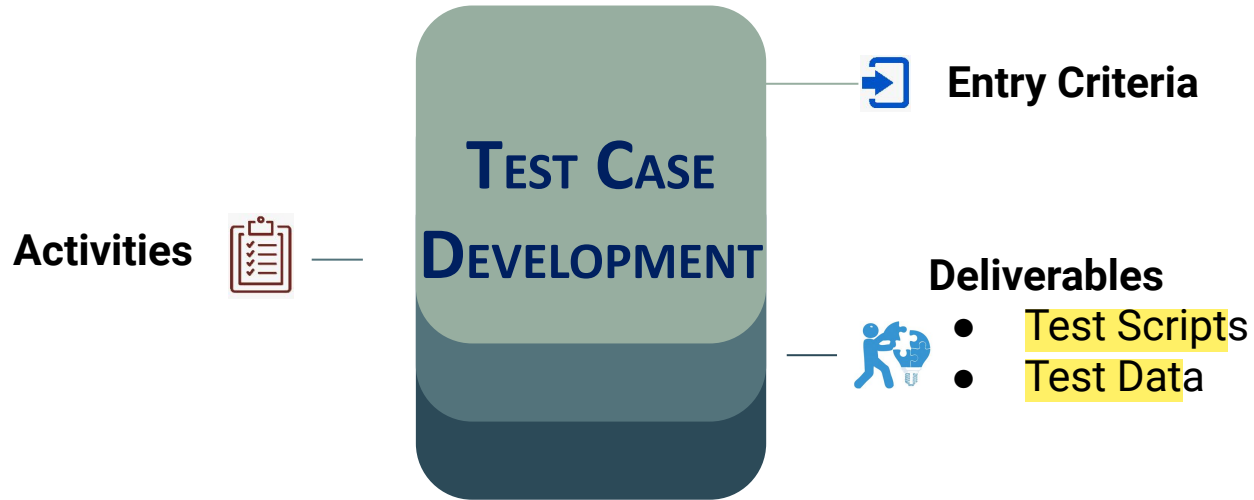
Environment Setup

<mark>Setup of software and hardware for the testing teams to execute test cases</mark>. It supports test execution with hardware, software and network configured.

Entry Criteria
- Environment setup is working as per the plan and checklist
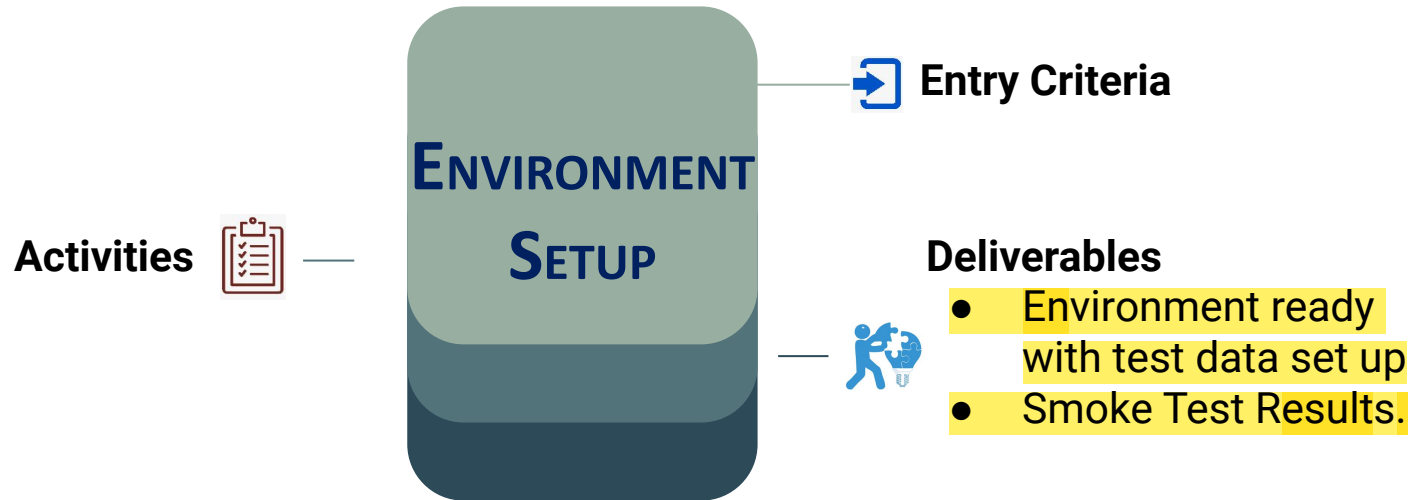- Test data setup is complete.
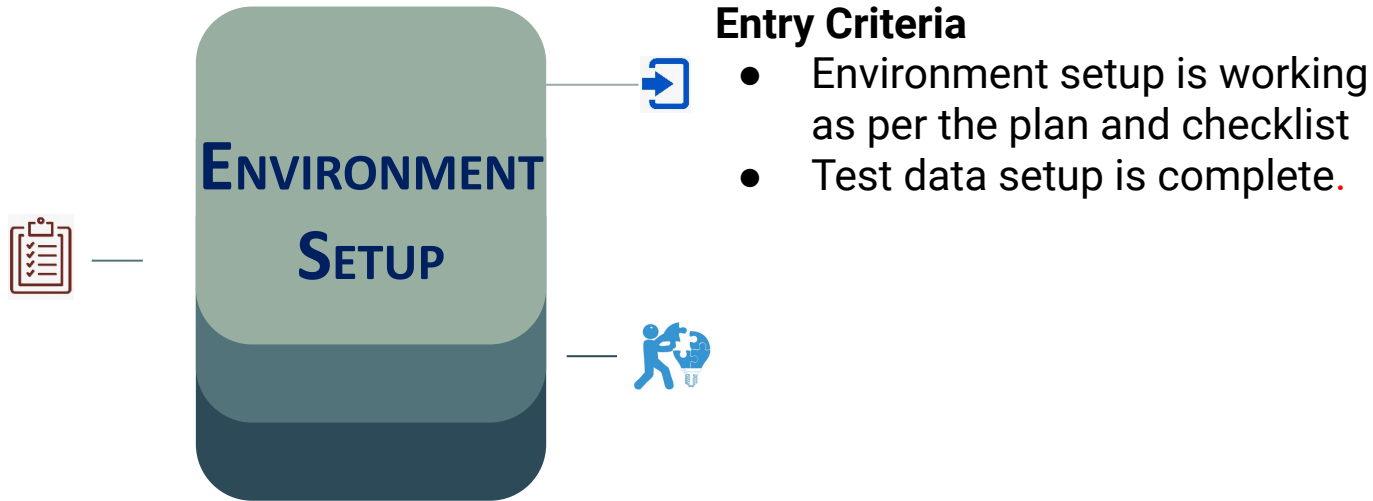
Environment Setup

Activities
- Understand the required architecture, environment set-up and prepare hardware and software requirement list for the Test Environment.
- Setup test Environment and test data
- Perform smoke test on the build

ENVIRONMENT SETUP

Entry Criteria

Environment Setup

ENVIRONMENT **S**ETUP

Entry Criteria

Activities

Deliverables
- Environment ready with test data set up
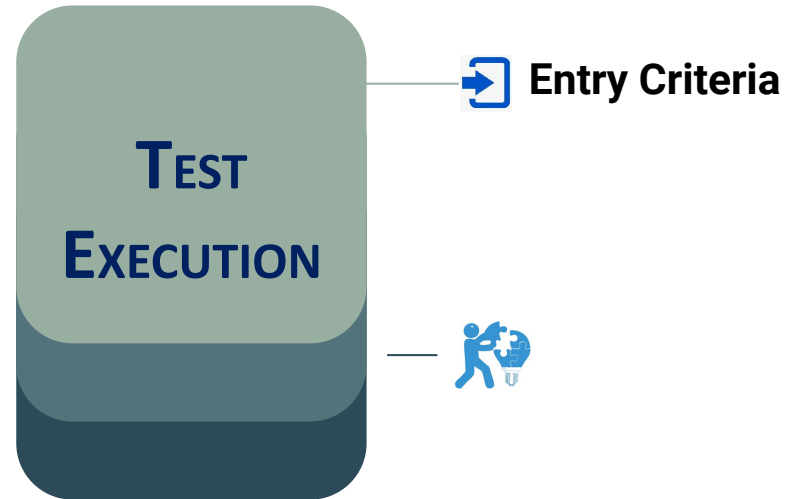- Smoke Test Results.

Test Execution

The process of executing the code and comparing the expected and actual results. When test execution begins, the test analysts start executing the test scripts based on test strategy allowed in the project.
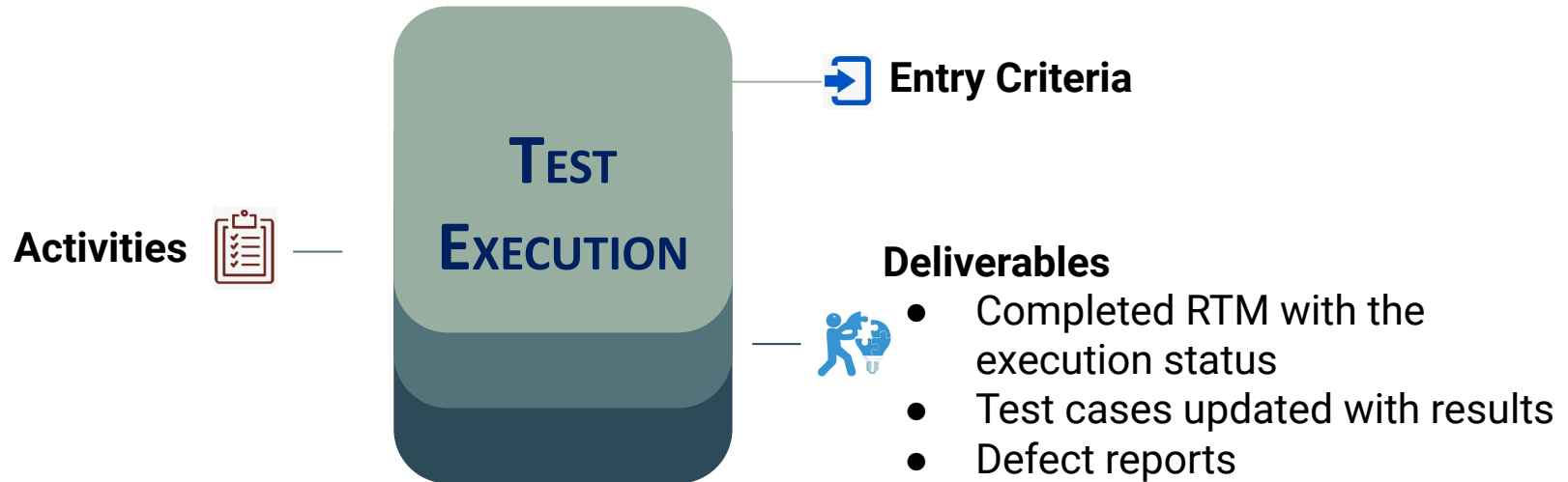
TEST EXECUTION

Entry Criteria
- Test cases
- Test scripts
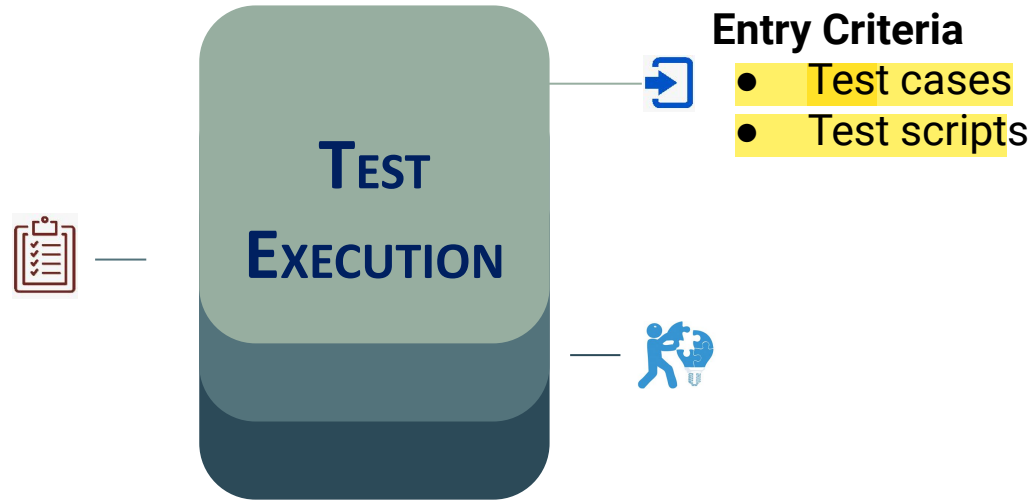
Test Execution

Activities
- Execute tests as per plan
- Document test results, and log defects for failed cases
- Map defects to test cases in RTM
- Retest the Defect fixes
- Track the defects to closure

TEST EXECUTION

→ **Entry Criteria**

Test Execution

TEST
EXECUTION

Activities

Entry Criteria

Deliverables
- Completed RTM with the execution status
- Test cases updated with results
- Defect reports

Test Closure

Involves calling out the testing team member meeting & evaluating cycle completion criteria based on Test coverage, Quality, Cost, Time, Critical Business Objectives, and Software.

TEST **C**LOSURE

Entry Criteria

- Test closure condition
- Test summary report
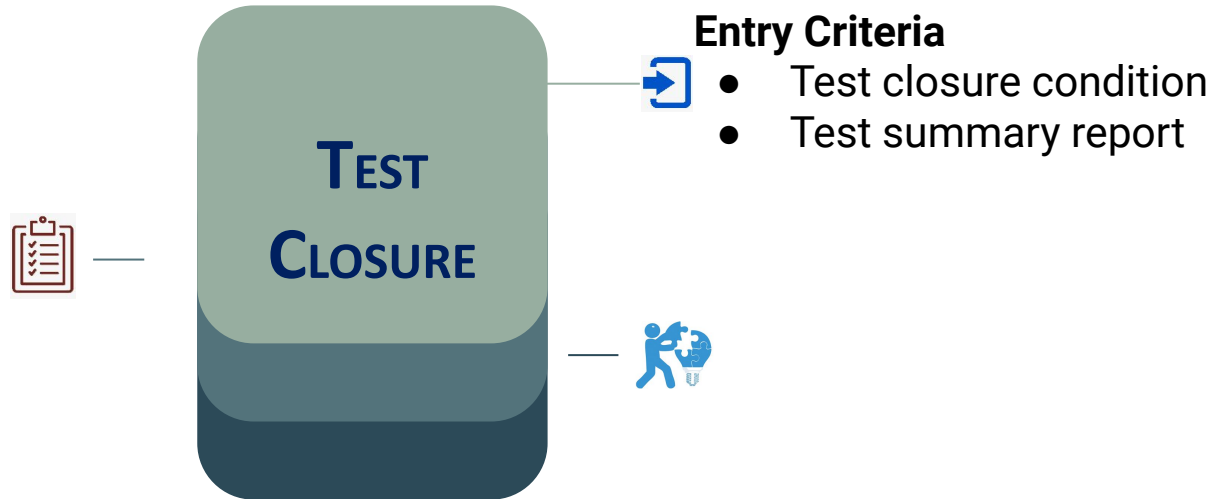
Test Closure

Activities
- Evaluate cycle completion criteria based on Time, Test coverage, Cost, Software, Critical Business Objectives, Quality
- Prepare test metrics based on the above parameters.
- Document the learning out of the project
- Prepare Test closure report
- Test result analysis to find out the defect distribution by type and severity.

TEST CLOSURE

→ **Entry Criteria**

Test Closure

Activities

TEST **C**LOSURE

→ **Entry Criteria**

Deliverables
- Test Closure report
- Test metrics
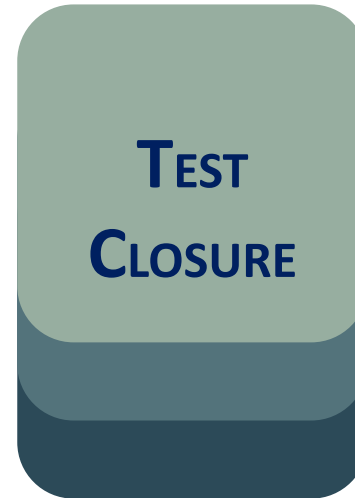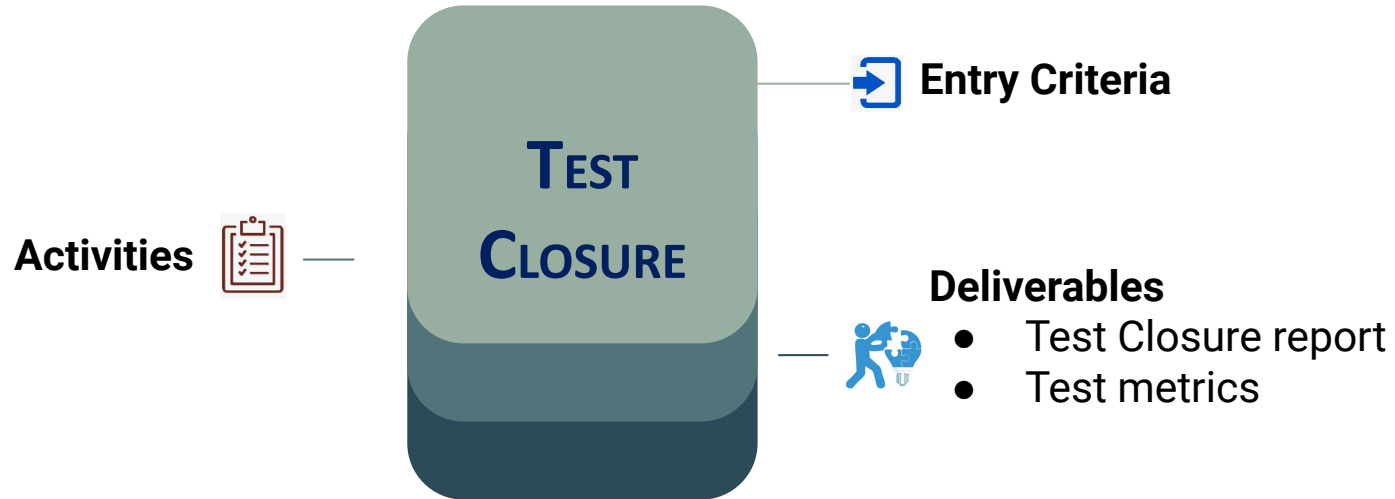
Appendix

REQUIREMENTS TRACEABILITY MATRIX					
Project Name: Online Flight Booking Application					
Business Requirements Document BRD		Functional Requirements Document FRD			Test Case Document
Business Requirement ID#	Business Requirement / Business Use case	Functional Requirement ID#	Functional Requirement / Use Case	Priority	Test Case ID#
BR_1	Reservation Module	FR_1	One Way Ticket booking	High	TC#001 TC#002
		FR_2	Round Way Ticket		TC#003 TC#004
		FR_3	Multicity Ticket booking	High	TC#005 TC#006
BR_2	Payment Module	FR_4	By Credit Card	High	TC#007 TC#008
		FR_5	By Debit Card	High	TC#009
		FR_6	By Reward Points	Medium	TC#010 TC#011