**Unit - 5: Levels of Testing**

**5.1 Test Planning-The goal of Test Planning:**

In software testing, documentation is very important. Testing should be documented to provide efficient resource control monitoring.

For successful testing, a test plan plays a very important role. Here, we will discuss the following points:

* **Introduction to Test Plan:** A test plan is a document that consists of all future testing-related activities
* **Importance of Test Plan:** It acts as a quick guide for the testing process.

It helps to avoid out-of-scope functionalities.

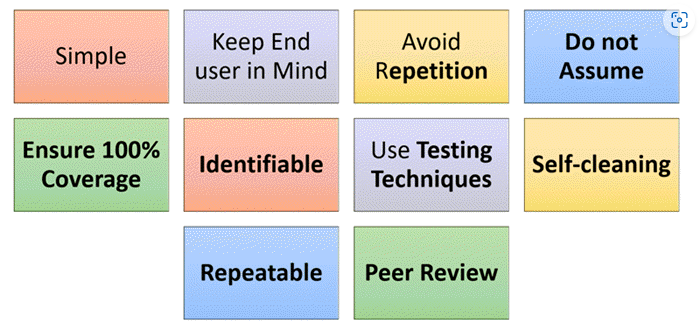
* **Test Plan Guidelines:** Avoid Overlapping and repetition. Avoid Lengthy Paragraph.

Use lists and tables.

* **Types of Test Plans:** includes multiple test strategies, emphasis on any one phase of testing, it is designed for specific types of testing.

**5.2 Test Planning Topics :**

* A test plan is a document that consists of all future testing-related activities.
* It is prepared at the project level and in general, it defines work products to be tested, how they will be tested, and test type distribution among the testers.



**5.3 Writing and Tracking Test Cases-The Goal of Test Case Planning :**

Test cases help guide the tester through a sequence of steps to validate whether a software application is free of bugs, and working as required by the end-user.

Learning how to write test cases for software requires basic writing skills, an attention to detail, and a good understanding of the application under test (AUT).

**How to write test cases for software:**

1> **Use a Strong Title:** “Login Page”

2> **Include a Strong Description:** The description should tell the tester what they’re going to test.

3> **Include Assumptions and Preconditions:** You should include any assumptions that apply to the test and any preconditions that must be met prior to the test being executed. 

4> **Keep the Test Steps Clear and Concise:** Test cases should be simple. Keep in mind, the person who wrote the test case might not be the same person who executes the test itself. don’t leave out any necessary details Write the test case so that anyone can go in and perform the test.

5> **Include the Expected result:** The expected result tells the tester what they should experience as a result of the test steps. This is how the tester determines if the test case is a “pass” or “fail”.

6> **Make it Reusable:** A good test case is reusable and provides long-term value to the software testing team.  When writing a test case, keep this in mind. You can save time down the road by re-using the test case instead of re-writing it.

**5.3 Sample of a Test Case :**

**What is a Test Case?**

In the context of software testing, a test case refers to the sequence of actions required to verify a specific feature or functionality. Essentially, the test case details the steps, data, prerequisites, and postconditions necessary to verify a feature.

It will lay out particular variables which QAs need to compare expected and actual results to conclude if the feature is working as it should. Test case components mention input, execution, and expected output/response. Basically, it tells engineers what to do, how to do it, and what results are acceptable.

Here is an example of a test case:

**Title**: Login Page – Authenticate Successfully on gmail.com

**Description:**A registered user should be able to successfully log in at gmail.com.

**Precondition:** the user must already be registered with an email address and password.

**Assumption:** a supported browser is being used.

**Test Steps:**

Navigate to gmail.com

In the ’email’ field, enter the email address of the registered user.

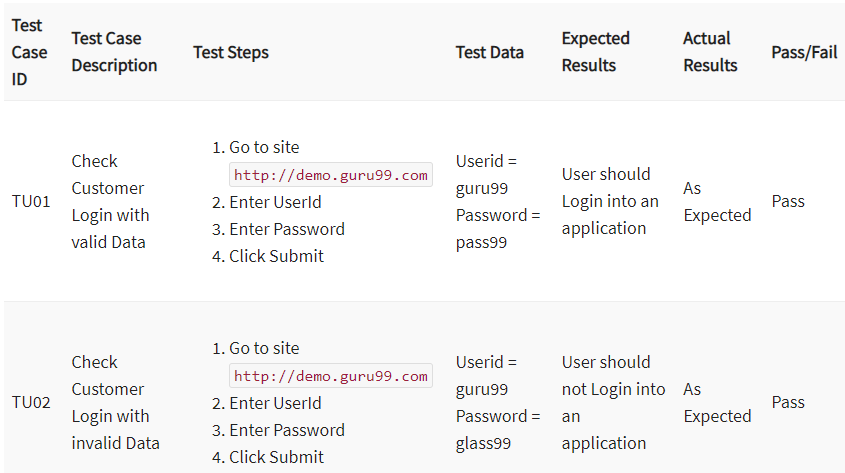
Click the ‘Next’ button.

Enter the password of the registered user

Click ‘Sign In’

**Expected Result:**A page displaying the Gmail user’s inbox should load, showing any new message at the top of the page.

**5.4 Test Case Planning Overview :**



**5.5 Test Case Organization and Tracking :**

One consideration that you should take into account when creating the test case documentation is how the information will be organized and tracked.

**Think about the questions that a tester or the test team should be able to answer:**

• Which test cases do you plan to run?

• How many test cases do you plan to run? How long will it take to run them? Test Case

Organization and Tracking

• Can you pick and choose test suites (groups of related test cases) to run on particular features or areas of the software?

• When you run the cases, will you be able to record which ones pass and which ones fail?

• Of the ones that failed, which ones also failed the last time you ran them?

• What percentage of the cases passed the last time you ran them?

**In your head**.

Don’t even consider this one, even for the simplest projects, unless you’re testing software for your own personal use and have no reason to track your testing.

You just can’t do it.

**Paper/documents:**

It's possible to manage the test cases for very small projects on paper. Tables and charts of checklists have been used effectively.

They're obviously a weak method for organizing and searching the data but they do offer one very important positive written checklist that includes a tester's initials or signature denoting that tests were run is excellent proof in a court-of-law that testing was performed.

**Spreadsheet:**

A popular and very workable method of tracking test cases is by using a spreadsheet.

By keeping all the details of the test cases in one place, a spreadsheet can provide an at-a-glance view of your testing status.

They're easy to use, relatively easy to set up, and provide good tracking and proof of testing.

**Custom database:**

The ideal method for tracking test cases is to use a Test Case Management Tool, a database programmed specifically to handle test cases.

Many commercially available applications are set up to perform just this specific task.

Microsoft Access, and many others provide almost drag-and-drop database creation that would let you build a database that mapped to the IEEE 829 standard in just a few hours.

You could then set up reports and queries that would allow you to answer just about any question regarding the test cases.

**5.6 Reporting Bugs- Getting Your Bugs Fixed:**

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.

**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 , more on this later

**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

**5.7 Isolating and Reproducing Bugs :**

You've just learned that to effectively report a bug, you need to describe it as obvious, general, and reproducible. In many cases this is easy. Suppose that you have a simple test case for a painting program that checks that all the possible colors can be used for drawing. If each and every time you select the color red the program draws in the color green, that's an obvious, general, and reproducible bug.

What would you do, though, if this incorrect color bug only occurs after you've run several of your other test cases and doesn't occur if you run the specific failing test case directly after rebooting the machine? What if it seems to occur randomly or only during a full moon? You'd have some sleuthing to do.

Isolating and reproducing bugs is where you get to put on your detective hat and try to figure out exactly what the steps are to narrow down the problem.

The good news is that there's no such thing as a random software bug if you create the exact same situation with the exact same inputs, the bug will reoccur.

The bad news is that identifying and setting up that exact situation and the exact same inputs can be tricky and time consuming. Once you know the answer, it looks easy. When you don't know the answer, it looks hard.

**Example:**

Some testers are naturally good at isolating and reproducing bugs. They can discover a bug and very quickly narrow down the specific steps and conditions that cause the problem.

For others, this skill comes with practice after finding and reporting many different types of bugs.

To be an effective software tester, though, these are skills that you'll need to master, so take every opportunity you can to work at isolating and reproducing bugs.