**Test Strategy:**

A Test Strategy document is a high-level document and normally developed by project manager. This document defines “Software Testing Approach” to achieve testing objectives. The Test Strategy is normally derived from the Business Requirement Specification document.

The Test Strategy document is a static document meaning that it is not updated too often. It sets the standards for testing processes and activities and other documents such as the Test Plan draws its contents from those standards set in the Test Strategy Document.

Some companies include the “Test Approach” or “Strategy” inside the Test Plan, which is fine and it is usually the case for small projects. However, for larger projects, there is one Test Strategy document and different number of Test Plans for each phase or level of testing.

**Components of the Test Strategy document**

* Scope and Objectives
* Business issues
* Roles and responsibilities
* Communication and status reporting
* Test deliverables
* Industry standards to follow
* Test automation and tools
* Testing measurements and metrices
* Risks and mitigation
* Defect reporting and tracking
* Change and configuration management
* Training plan

**Test Plan:**

The Test Plan document on the other hand, is derived from the Product Description, Software Requirement Specification SRS, or Use Case Documents.  
The Test Plan document is usually prepared by the Test Lead or Test Manager and the focus of the document is to describe what to test, how to test, when to test and who will do what test.

It is not uncommon to have one Master Test Plan which is a common document for the test phases and each test phase have their own Test Plan documents.

There is much debate, as to whether the Test Plan document should also be a static document like the Test Strategy document mentioned above or should it be updated every often to reflect changes according to the direction of the project and activities.

My own personal view is that when a testing phase starts and the Test Manager is “controlling” the activities, the test plan should be updated to reflect any deviation from the original plan. After all, Planning and Control are continuous activities in the formal test process.

**Components of the Test Plan document**

* Test Plan id
* Introduction
* Test items
* Features to be tested
* Features not to be tested
* Test techniques
* Testing tasks
* Suspension criteria
* Features pass or fail criteria
* Test environment (Entry criteria, Exit criteria)
* Test deliverables
* Staff and training needs
* Responsibilities
* Schedule

### What is a Test Case?

A Test Case is defined as a set of actions executed to verify a particular feature or functionality of the software application. A test case is an indispensable component of the Software Testing LifeCycle that helps validate the AUT (Application Under Test).

### Test Scenario Vs Test Case

Test scenarios are rather vague and cover a wide range of possibilities. Testing is all about being very specific.

For a Test Scenario: Check Login Functionality there many possible test cases are:

* Test Case 1: Check results on entering valid User Id & Password
* Test Case 2: Check results on entering Invalid User ID & Password
* Test Case 3: Check response when a User ID is Empty & Login Button is pressed, and many more

Below is a format of a standard login Test case

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test Case ID** | **Test Scenario** | **Test Steps** | **Test Data** | **Expected Results** | **Actual Results** | **Pass/Fail** |

## Best Practice for writing good Test Case Example.

**1. Test Cases need to be simple and transparent:**

**2. Create Test Case with End User in Mind**

**3. Avoid test case repetition.**

**4. Do not Assume**

**5. Ensure 100% Coverage**

**6. Test Cases must be identifiable.**

**7. Implement Testing Techniques**

8. **Self-cleaning**

9. **Repeatable** **and self-standing**

**10. Peer Review.**

**Preparing a Test Case Document:**

1. **Equivalence Partitioning (EP):**

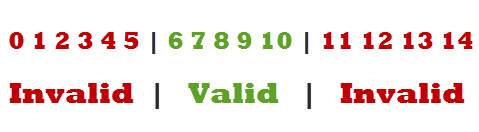
Equivalence partitioning is a Test Case Design Technique to divide the input data of software into different equivalence data classes. Test cases are designed for equivalence data class. In this technique, only one condition to be tested from each partition. Because we assume that, all the conditions in one partition behave in the same manner by the software.

**Example:**

A text field permits only numeric characters

Length must be 6-10 characters’ long

Partition according to the requirement should be like this

[](https://e8c5h7a9.stackpathcdn.com/wp-content/uploads/2013/11/example-for-equivalence-class-partitioning.jpg)

1. **Boundary Value Analysis (BVA):**

Boundary value analysis is a test case design technique to test boundary value between partitions (both valid boundary partition and invalid boundary partition). A boundary value is an input or output value on the border of an equivalence partition, includes minimum and maximum values at inside and outside boundaries. Normally Boundary value analysis is part of stress and negative testing.

**Example:**

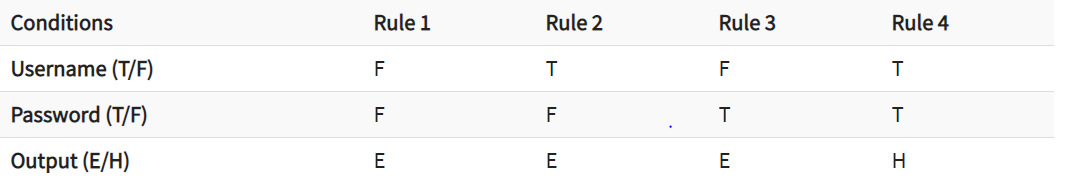
Suppose you have very important tool at office, accepts valid User Name and Password field to work on that tool, and accepts minimum 8 characters and maximum 12 characters. Valid range 8-12, Invalid range 7 or less than 7 and Invalid range 13 or more than 13.

[](https://e8c5h7a9.stackpathcdn.com/wp-content/uploads/2013/11/example-for-boundary-value-analysis1.png)

1. **Decision Table Testing:**

Decision table testing is a software testing technique used to test system behaviour for different input combinations. This is a systematic approach where the different input combinations and their corresponding system behaviour (Output) are captured in a tabular form. That is why it is also called as a **Cause-Effect** table where Cause and effects are captured for better test coverage

**Example:** Decision table for Login Screen



* T – Correct username/password
* F – Wrong username/password
* E – Error message is displayed
* H – Home screen is displayed

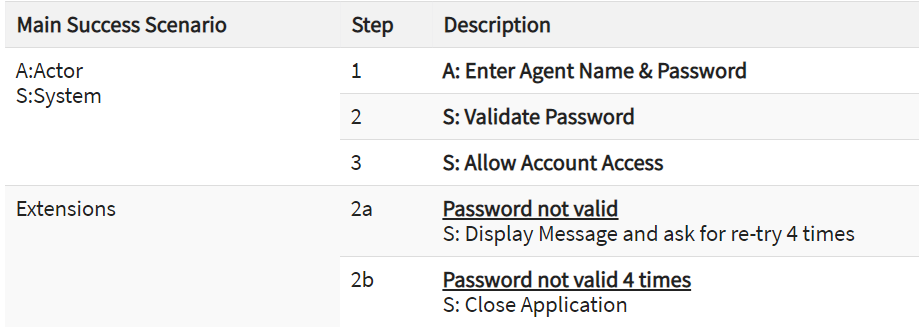
1. **Use Case Testing:**

A use case is a description of a particular use of the system by an actor or user. It is used widely in developing tests at system or acceptance level

Use Case Testing is defined as a software testing technique, that helps identify test cases that cover the entire system, on a transaction by transaction basis from start to the finishing point.

**Example:**

In a use-case, an actor is represented by "A" and system by "S". We create Use for a login functionality of a Web Application as shown below



* Consider the first step of an end to end scenario for a login functionality for our web application where the Actor enters email and password.
* In the next step, the system will validate the password
* Next, if the password is correct, the access will be granted
* There can be an extension of this use case. In case password is not valid system will display a message and ask for re-try four times
* If Password, not valid four times system will ban the IP address.

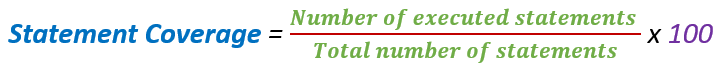
Here we will test the success scenario and one case of each extension.

This is USE-Case testing in Software Engineering

1. **Statement Coverage:**

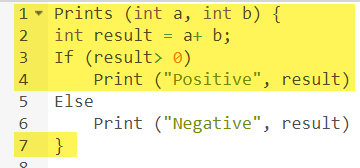
Statement coverage is a white box test design technique which involves execution of all the executable statements in the source code at least once. It is used to calculate and measure the number of statements in the source code which can be executed given the requirements.

Statement coverage is used to derive scenario based upon the structure of the code under test.

[](https://www.guru99.com/images/jsp/030116_0814_LearnStatem1.png)

**Scenario 1:**

If A = 3, B = 9

[](https://www.guru99.com/images/1/102518_1122_CodeCoverag2.png)

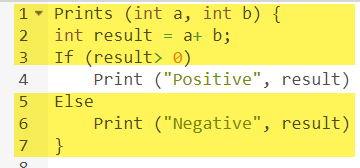
The statements marked in yellow color are those which are executed as per the scenario

Number of executed statements = 5, Total number of statements = 7

Statement Coverage: 5/7 = 71%

**Scenario 2:**

If A = -3, B = -9

[](https://www.guru99.com/images/1/102518_1122_CodeCoverag4.png)

The statements marked in yellow color are those which are executed as per the scenario.

Number of executed statements = 6, Total number of statements = 7

Statement Coverage: 6/7 = 85%

But overall if you see, all the statements are being covered by 2nd scenario's considered. So we can conclude that overall statement coverage is 100%.

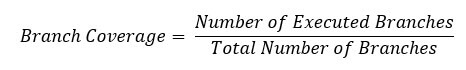
1. **Branch Coverage:**

In the branch coverage, every outcome from a code module is tested. For example, if the outcomes are binary, you need to test both True and False outcomes.

It helps you to ensure that every possible branch from each decision condition is executed at least a single time.

By using Branch coverage method, you can also measure the fraction of independent code segments. It also helps you to find out which is sections of code don't have any branches.

The formula to calculate Branch Coverage:

[](https://www.guru99.com/images/1/102518_1122_CodeCoverag13.jpg)

