# What is a Test Case

A test case is a set of conditions or variables under which a tester will determine whether a system under test satisfies requirements or works correctly.

The process of developing test cases can also help find problems in the requirements or design of an application.

# Test Case Design Techniques

## First, what is the parameters of a test case?

- **ID**: auto-generated by the tool
- **Title:** contains a brief description of the idea of the test case.
- **Prerequisite:** Identify all setup information needed for running the test. Setup information includes testing prerequisite such as data preparation, hardware (mobile applications), software and browsers. Here you may need to mention that there may be an order of execution whereby a test case may depend upon another test case running successfully and leaving the system in a state so that the second test case can successfully be executed.
- **Test case Description:** purpose of the test. It should allow someone with no prior knowledge of the test case to get a clear understanding of what is being covered without going through all of the test steps.
- **Test Steps:** actions/steps (by user) and expected results (by the system).
- **Expected Results:** included above
- **Actual Results**: defined in the time of running the test
- **Attachments:** to add any reference like a graphic design if needed
- **Scenario Type:** this is a custom field added to the test case form to define if it is a positive or negative scenario (used in the QC process).
- **Status:** should be updated upon test case execution

## Test Case Design completeness

When a test case design task is considered complete?

1. Test cases must cover all requirements – any single statement in the requirements document (RSD) should have a corresponding set of test cases (Positive & negative TCs)
2. Test cases should include all the correct steps and their corresponding results that the tester will examine during execution.
3. Test case design should cover as many Negative test cases as possible. (Referring to test case design techniques).
4. A Test case must not be just a copy from the requirements document statements.
5. Steps and Expected Results must be clear so that anybody can understand and execute.

6. It must be clear whether the step written is done by the system or by the user.
7. Test cases should be clear from grammatical or spelling mistakes.

## Test Case Design Techniques:

It is not possible to perform exhaustive testing for each set of test data, especially when there is a large pool of input combinations.

We need an easy way or special techniques that can select test cases intelligently from the pool of test-case, such that all possible scenarios are covered.

We usually use two techniques - **Equivalence Partitioning & Boundary Value Analysis testing techniques** to achieve this

### 1. Boundary value analysis (BVA)

Boundary testing is the process of testing between extreme ends or boundaries between partitions of the input values.

So these extreme ends like Start- End, Lower- Upper, Maximum-Minimum, Just Inside-Just Outside values are called boundary values and the testing is called "boundary testing".
The basic idea in boundary value testing is to select input variable values at their:

- Minimum
- Just above the minimum
- A nominal value
- Just below the maximum
- Maximum

### 2. Equivalence Partitioning

Equivalent Class Partitioning is a black box technique which can be applied to all levels of testing like unit, integration, system, etc. In this technique, you divide the set of test condition into a partition that can be considered the same.

It divides the input data of software into different set that expected to make the same behavior.
You can apply this technique, where there is a range in input field.

**Example:**
In a flight reservation form, “Number of Tickets” field should accept only values from 1 to 10 and ticket is booked. While value 11 to 99 are considered invalid for reservation and error message will appear, "Only ten tickets may be ordered at one time."

We cannot test all the possible values because if done, the number of test cases will be more than 100
Here we have 4 equivalent sets:

- Any Number greater than 10 entered in the reservation column (let say 11) is considered invalid.
- Any Number less than 1 that is 0 or below, then it is considered invalid.
- Numbers 1 to 10 are considered valid

The divided sets are called Equivalence Partitions or Equivalence Classes. Then we pick only one value from each partition for testing. The hypothesis behind this technique is that **if one condition/value in a partition passes all others will also pass**. Likewise, **if one condition in a partition fails, all other conditions in that partition will fail**

**Boundary Value Analysis**- in Boundary Value Analysis, you test boundaries between equivalence partitions

In our earlier example instead of checking, one value for each partition you will check the values at the partitions like 0, 1, 10, 11 and so on. As you may observe, you test values at both valid and invalid boundaries. Boundary Value Analysis is also called range checking.

## 3. Decision Table

Decision table testing is a testing technique used to test system behavior for different input combinations. This is a systematic approach where the different input combinations and their corresponding system behavior (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.

This testing technique becomes important when it is required to test different combination. It also helps in better test coverage for complex business logic.

A Decision Table is a tabular representation of inputs versus rules/cases/test conditions. Let's learn with an example.

**Example 1:** Decision Base Table for Login Screen

The condition is simple if the user provides correct username and password the user will be redirected to the homepage. If any of the input is wrong, an error message will be displayed.

So we have here 2 input conditions (Username is correct and Password is correct) and 2 rules (it might be True or False).
Here is the possible combination of rules that could happen. Each column is a separate test case.

| Conditions | Rule 1 | Rule 2 | Rule 3 | Rule 4 |
|---|---|---|---|---|
| Username is correct | F | T | F | T |
| Password is correct | F | F | T | T |
| Expected Output | Error msg | Error msg | Error msg | Homepage |

Interpretation:

Case 1 – Username and password both were wrong. The user is shown an error message.

Case 2 – Username was correct, but the password was wrong. The user is shown an error message.

Case 3 – Username was wrong, but the password was correct. The user is shown an error message.

Case 4 – Username and password both were correct, and the user navigated to homepage

**Example 2:** Decision Base Table for Upload Screen

Now consider a dialogue box which will ask the user to upload photo with certain conditions like –

1. You can upload only '.jpg' format image
2. file size less than 32kb
3. Resolution 137*177.

If any of the conditions fails, the system will throw an error message stating the issue and if all conditions are met photo will be updated successfully

| Conditions | Case 1 | Case 2 | Case 3 | Case 4 | Case 5 | Case 6 | Case 7 | Case 8 |
|---|---|---|---|---|---|---|---|---|
| Format | .jpg | .jpg | .jpg | .jpg | Not .jpg | Not .jpg | Not .jpg | Not .jpg |
| Size | Less than 32kb (ex. 31 kb) | Less than 32kb | More than 32kb | More than 32kb (ex. 33 kb) | Less than 32kb | Less than 32kb | More than 32kb | More than 32kb |
| resolution | 137*177 | Not 137*177 | 137*177 | Not 137*177 | 137*177 | Not 137*177 | 137*177 | Not 137*177 |
| Output | Photo uploaded | Error msg: resolution mismatch | Error msg: size mismatch | Error: size and resolution mismatch | Error: format mismatch | Error: format and resolution mismatch | Error: format and size mismatch | Error: format, size, & resolution mismatch |

## 4. State Transition Testing

State Transition testing is defined as the testing technique in which changes in input conditions cause's state changes in the Application under Test.

It is a black box testing technique in which the tester analyzes the behavior of an application under test for different input conditions in a sequence. In this technique, tester provides both positive and negative input test values and record the system behavior.

It is the model on which the system and the tests are based. Any system where you get a different output for the same input, depending on what has happened before, is a finite state system.

State Transition Testing Technique is helpful where you need to test different system transitions.

When to Use State Transition?

- This can be used when a tester is testing the application for a finite set of input values.
- When the tester is trying to test sequence of events that occur in the application under test. I.e., this will allow the tester to test the application behavior for a sequence of input values.
- When the system under test has a dependency on the events/values in the past.

There are 4 main components of the State Transition Model as below

1) States that the software might get

2) Transition from one state to another

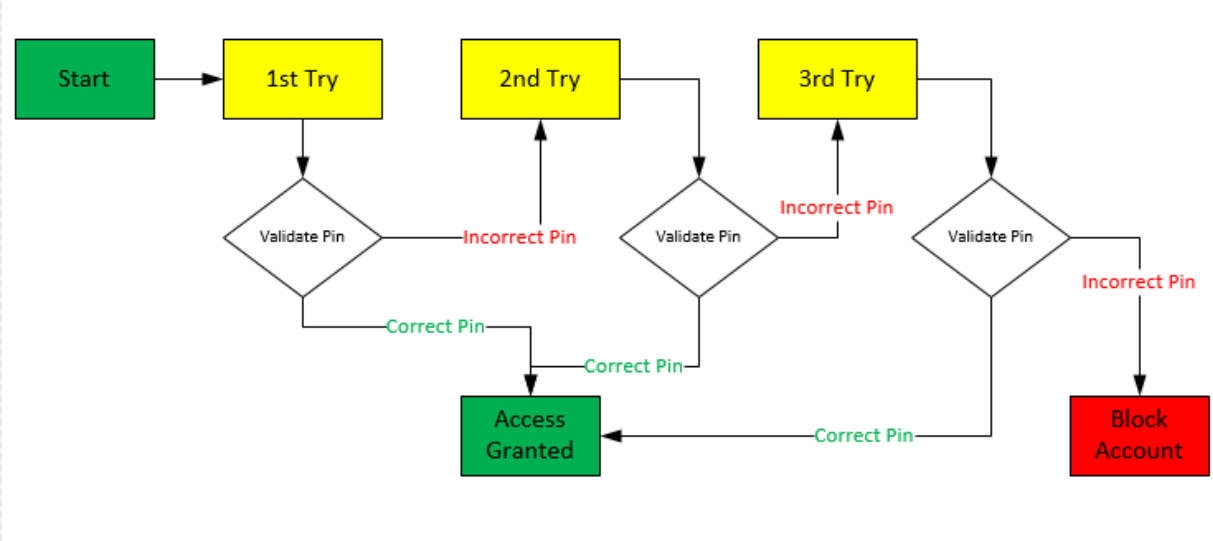
3) Events that origin a transition like closing a file or withdrawing money

4) Actions that result from a transition (an error message or being given the cash.)

**Example 1:**

Let's consider an ATM system function where if the user enters the invalid password three times the account will be locked.

In this system, if the user enters a valid password in any of the first three attempts the user will be logged in successfully. If the user enters the invalid password in the first or second, try the user will be asked to re-enter the password. And finally, if the user enters incorrect password 3rd time, the account will be blocked.




In the diagram whenever the user enters the correct PIN he is moved to Access granted state, and if he enters the wrong password he is moved to next try and if he does the same for the 3rd time the account blocked state is reached.

**State Transition Table:**

| | Correct PIN | Incorrect PIN |
|---|---|---|
| **S1) Start** | | |
| **S2) 1st attempt** | **S5** | **S3** |
| **S3) 2nd attempt** | **S5** | **S4** |
| **S4) 3rd attempt** | **S5** | **S6** |
| **S5) Access Granted** | - | - |
| **S6) Account blocked** | - | - |

Here we have 6 test cases:

1. If the user entered a correct pin in the 1st attempt, he'll grant access to the system
2. If the user entered a correct pin in the 2st attempt, he'll grant access to the system
3. If the user entered a correct pin in the 3st attempt, he'll grant access to the system
4. If the user entered an incorrect pin in the 1st attempt, he'll redirect to the 2nd attempt
5. If the user entered an incorrect pin in the 2st attempt, he'll redirect to the 3rd attempt
6. If the user entered an incorrect pin in the 3st attempt, his account will be blocked

## Core Principles

1. **Test case is an important output of the testing team;**
2. The test cases I am writing could be executed by others, this doesn't happen most of the time. There are three possible audiences for the test cases you write:
    a. Testers other than myself
    b. Developers- it looks really bad when we give them test cases to execute and they discover that they are not adding any value for them or test cases which are invalid.
    c. Business users. Your test cases will be used in preparing the UAT document.

    Think of your audience, ask yourself if all the information needed to run the test are included in the test case.
3. **It should make our lives easier when we write down a clear set of test cases with complete ideas. We will guarantee test coverage.**
4. Unless you are convinced of that, test case design task is just going to be a tedious task.
5. It's very important to keep our test cases status updated during execution. This helps in tracking out testing progress.
6. We need to know why we are creating test cases. For two reasons:
    a. To validate that possible business scenarios will work as expected. Normal and abnormal scenarios:
    Example: 2 Employees working on the same tasks pool, both are submitting action on the same task in the same time by coincidence. (we name this **Concurrency** scenario)
    b. The user trials to break the system will fail in real life:
    Examples**:** a user is submitting a vacation request from 2 different browser tabs in order to take vacation that is exceeding his balance. This will work if the balance validation is done on client side only.

7. If requirements are updated after you're done with your TC design, we should schedule time for test cases update; if you face a problem with managers to allocate time for aht ask leads to fight for you
8. Invest more time to think about negative scenarios (special sequence ones)
9. Invest more time to think about integration scenarios (among multiple modules)

# Assignment

## Question:

**Please apply what you've learnt here on the following examples:**
**An update is going to be done on a national aviation portal to add an offer on the tickets submitted through this portal. It is for the occasion of their anniversary. It's a 20% offer that applies only in case the following conditions are met:**

1. **The user is buying the ticket from Cairo to any international country**
2. **The user is submitting his booking request during this period (from 15 Dec 2024 to 18 Dec 2024)**
3. **The flight date is in this period (29 Dec 2024 to 3 Jan 2025), and return date doesn't matter.**