**TESTING TECHNIQUES**

**SOFTWARE TESTING TECHNIQUES**Software Testing Techniques help you design better test cases. Since exhaustive testing is not possible; Manual Testing Techniques help reduce the number of test cases to be executed while increasing test coverage. They help identify test conditions that are otherwise difficult to recognize.

Below mentioned are the techniques:

* [Boundary Value Analysis (BVA)](https://www.guru99.com/software-testing-techniques.html#1)
* [Equivalence Class Partitioning](https://www.guru99.com/software-testing-techniques.html#2)
* [Decision Table based testing.](https://www.guru99.com/software-testing-techniques.html#3)
* [State Transition](https://www.guru99.com/software-testing-techniques.html#4)
* [Error Guessing](https://www.guru99.com/software-testing-techniques.html#5)

**Boundary Value Analysis (BVA)**

Boundary value analysis is based on testing at the boundaries between partitions. It includes maximum, minimum, inside or outside boundaries, typical values and error values.

It is generally seen that a large number of errors occur at the boundaries of the defined input values rather than the center. It is also known as BVA and gives a selection of test cases which exercise bounding values.

25Ma TEST CASE Software Testing Tutorial

This black box testing technique complements equivalence partitioning. This software testing technique base on the principle that, if a system works well for these particular values then it will work perfectly well for all values which comes between the two boundary values.

**Guidelines for Boundary Value analysis**

* If an input condition is restricted between values x and y, then the test cases should be designed with values x and y as well as values which are above and below x and y.
* If an input condition is a large number of values, the test case should be developed which need to exercise the minimum and maximum numbers. Here, values above and below the minimum and maximum values are also tested.
* Apply guidelines 1 and 2 to output conditions. It gives an output which reflects the minimum and the maximum values expected. It also tests the below or above values.

**Example:**

Input condition is valid between 1 to 10

Boundary values 0,1,2 and 9,10,11

**Equivalence Class Partitioning**

Equivalent Class Partitioning allows you to divide set of test condition into a partition which should be considered the same. This software testing method divides the input domain of a program into classes of data from which test cases should be designed.

The concept behind this technique is that test case of a representative value of each class is equal to a test of any other value of the same class. It allows you to Identify valid as well as invalid equivalence classes.

**Example:**

Input conditions are valid between

Consider a condition where you need to test Age field in a form which has valid input for 16-70 for age

**Graphical user interface, diagram

Description automatically generated with medium confidence**

Valid Partitions are values that should be accepted by the component or system under test. This partition is called “**Valid Equivalence Partition.**”

Invalid Partitions are values that should be rejected by the component or system under test. This partition is called “**Invalid Equivalence Partition.**”

Our Test conditions in Such case could be:

* ***Enter Age = 5***
* ***Enter Age = 30***
* ***Enter Age = 75***

*You can sub divide the input for valid conditions range 16-70 [Considering 16-30 as one set and 31-70 as another set]*

**Decision Table Based Testing.**

A decision table is also known as to Cause-Effect table. This software testing technique is used for functions which respond to a combination of inputs or events. For example, a submit button should be enabled if the user has entered all required fields.

The first task is to identify functionalities where the output depends on a combination of inputs. If there are large input set of combinations, then divide it into smaller subsets which are helpful for managing a decision table.

For every function, you need to create a table and list down all types of combinations of inputs and its respective outputs. This helps to identify a condition that is overlooked by the tester.

**Following are steps to create a decision table:**

* Enlist the inputs in rows.
* Enter all the rules in the column.
* Fill the table with the different combination of inputs.
* In the last row, note down the output against the input combination.

**Example**: A submit button in a contact form is enabled only when all the inputs are entered by the end user.

A screenshot of a computer

Description automatically generated with medium confidence

**State Transition**

In State Transition technique changes in input conditions change the state of the Application Under Test (AUT). This testing technique allows the tester to test the behavior of an AUT. The tester can perform this action by entering various input conditions in a sequence. In State transition technique, the testing team provides positive as well as negative input test values for evaluating the system behavior.

**Guideline for State Transition:**

* State transition should be used when a testing team is testing the application for a limited set of input values.
* The technique should be used when the testing team wants to test sequence of events which happen in the application under test.

**Example:**

In the following example, if the user enters a valid password in any of the first three attempts the user will be able to log in successfully. If the user enters the invalid password in the first or second try, the user will be prompted to re-enter the password. When the user enters password incorrectly 3rd time, the action has taken, and the account will be blocked.

**State transition diagram**

**Diagram

Description automatically generated**

In this diagram when the user gives the correct PIN number, he or she is moved to Access granted state. Following Table is created based on the diagram above-

**State Transition Table**

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

In the above-given table when the user enters the correct PIN, the state is transitioned to Access granted. And if the user enters an incorrect password, he or she is moved to next state. If he does the same 3rd time, he will reach the account blocked state.

**Error Guessing**

**Error Guessing** is a software testing technique based on guessing the error which can prevail in the code. The technique is heavily based on the experience where the test analysts use their experience to guess the problematic part of the testing application. Hence, the test analysts must be skilled and experienced for better error guessing.

The technique counts a list of possible errors or error-prone situations. Then tester writes a test case to expose those errors. To design test cases based on this software testing technique, the analyst can use the past experiences to identify the conditions.

**Guidelines for Error Guessing:**

* The test should use the previous experience of testing similar applications.
* Understanding of the system under test.
* Knowledge of typical implementation errors.
* Remember previously troubled areas.
* Evaluate Historical data & Test results.

**Conclusion**

* Software testing Techniques allow you to design better cases. There are five primarily used techniques.
* Boundary value analysis is testing at the boundaries between partitions.
* Equivalent Class Partitioning allows you to divide set of test condition into a partition which should be considered the same.
* Decision Table software testing technique is used for functions which respond to a combination of inputs or events.
* In State Transition technique changes in input conditions change the state of the Application Under Test (AUT)
* Error guessing is a software testing technique which is based on guessing the error which can prevail in the code.