# Test Design

**Test Scenarios & Test Cases**

****

**Test Scenario**

* Possible areas to be tested (or) what is to be tested.
* In Agile testing Scenario defines what to test in userstory
* In general test scenarios listed in Test Plan document.
* Test scenarios identified by the testlead or Sr. Test Engineer.
* **Entry Criteria for identifying test Scenarios:**
* Approved Test Plan
* Approved SRS
* Test Scenario template
* Any design documents available (if any)
* Any blueprints available (if any)

**Exit Criteria for test Scenarios:**

* Test scenarios Should reviewed & Approved.
* Once test Scenarios are approved test lead will create a baseline for test Scenarios (TS1.0v) and update the scenarios into Common Repository.

**What is a Test Case.?**

* A test case is set of user actions and sub sequent response from the system
* Every positive and negative action at software what sub sequent response receive from system we design as Test-case document.
* Here tester prepare testcases as checklist when software is under development, Once software is ready tester use same checklist to verify is software developed as client expected.

**Entry Criteria to prepare test cases:**

* Approved Test Plan
* Approved SRS
* Approved FRS [or] Use case template[Use case template is ideal way to design testcase]
* Approved Test Scenarios
* Test Case template

**Exit Criteria for test cases:**

* Test Cases should be reviewed & Approved.

**Test Case contains**

A test case should contain particulars such as:

* Objective
* Test Conditions
* Input data
* Expected results
* Location of the files to be used in Test Case [Test scenario reference numbers, Document Reference numbers]
* Trouble shooting guidelines

**Good Test Case Design**

A good test case satisfies the following criteria

* Effective – Finds Faults (Objective)
* Evaluable – Easy to maintain
* Requirement Completeness
* Easy to understand
* Should not be out of scope - do not contain un necessary things
* Don’t predict stick to documentation
* Testcases should write in uniform way. [If you started test cases for one requirement don’t merge other requirement ]

**Test Case Design Techniques**

**Why test techniques**

* Exhaustive testing (Use of all possible inputs and conditions) is impractical.
* Need thought processes that help us to select test cases more intelligently
* Test case design techniques helps to cover more areas to test software.

**Advantages of techniques**

* Different people: Similar probability of finding faults
* Effective testing: to find more deviations (faults)
* One can focus or pay more attention on specific types of faults
* To know you are testing right thing.
* Avoid duplication
* Identify the best possible combinations to cover maximum conditions

**Test Case design techniques can be broadly categorized into 2:**

* Black Box (Functional) Test case design Techniques
* White Box (Structural) Test case design Techniques
* Experience Testing [Informal testing]

**Black Box Test case design Techniques:**

* Boundary value Analysis
* Equivalent class Partitioning
* Decision table
* Error guessing
* State transition technique

**Boundary Value Analysis**

* In this method tester has to concentrate more on the boundaries of the input values.
* In testing boundary conditions have a higher probability of detecting errors
* Here tester verify the faults at near boundaries

**Example:**

* Consider any edit field which can accept values b/w 18-35.
* According to BVA method the valid inputs are:
* 17,18,….35 & 36
* If the range is 'a' to 'b'
* Then valid inputs are:  a-1, a, a+1, b-1, b, &b+1

**Example:🡪**

**Consider Zipcode editbox accept Max number 6 only.**

**According to BVA testable boundaries are**

* **Max, Max+1, Max-1**
* **6,7,5**

**Example**

* Program Accepts 1 To 100 Characters and identify Boundary values
* The valid inputs are:  a-1, a, a+1, b-1, b, & b+1
* 0, 1 to 100, 101



**Equivalence Partitioning**

* Equivalence partitioning is a method for deriving test cases

In this method given input is divided into number of equivalence classes.

* From each equivalence class one input value is choose for testing.
* Equivalent partitioning drastically cuts down the number of test cases
* We can Label the ECP Classes as “Valid” and Invalid”

**Example:**

Consider any edit field which can accept values b/w 4-12.

From the given range of i/p value we can form 3 equivalence classes.

     i) Less than 4.

   ii) Between 4 -12.

  iii) Greater than 12.

* Consider edit field accept only Alphabets
  + Valid :🡪 Alphabets [a-z][A\_Z]
  + Invalid:🡪Numerics, Special Characters, Spaces

Consider edit field accept only Alphabets and Numerics, At least One Numeric value

* + Valid:🡪 Alphabets[a-z] + Numerics [1]
  + Invalid:🡪 only Numerics, Only Alphabets, Special Characters, Spaces, Blank
* Consider edit fields accept only alphabets, Should starts with uppercase and should contains at least one special characters

Valid:🡪 Alphabets Starts with Uppercase characters, Alphabets with One Special character

Invalid:🡪 Alphabets Starts with lowercase, Alphabets without special character, Numbers, Spaces, Starts with

Numbers, Starts with special characters

|  |  |  |
| --- | --- | --- |
| **User Name** | **Role** | **Rights** |
| Abhay | Financial Prof. | Assign |
| Sriram | Customer | View, Add, Update |
| Ravi | Customer | View, Add, Update |
| Dinakar | Financial Prof. | Assign |
| Bhaskar | Employee | View, Delete, Add, Update |
| Ramya | Financial Prof. | Assign |
| Pranathi | Financial Prof. | Assign |
| Harsha | Employee | View, Delete, Add, Update |
| Suchitra | Agent | View |

* Identify similar functionalities and divide equally
* Divide the inputs, outputs, etc., into areas which are same
* Here the assumption is, if one value works then all should work
* Identify one from each partition instead of all from one

|  |  |  |
| --- | --- | --- |
| **Role** | **User Name** | **Rights** |
| Financial Prof. | Abhay | Assign |
| Customer | Sriram | View, Add, Update |
| Employee | Bhaskar | View, Delete, Add, Update |
| Agent | Suchitra | View |

**Example:**

Specifications state that a max of 4 purchase orders can be registered against any one product:

The equivalence classes are:

Valid Class: 1 – 4 Purchase Orders

Invalid Classes: # Purchase Orders > 4, # Purchase Orders < 1

**Example:**

* Customer Name – 2 to 64 Chars
* Account Number – 6 digits, first digit is non zero
* Loan Amt requested – $500 to $ 9000
* Loan Term – 1 to 30 years
* Monthly Repayment – Min of $ 10

**Customer Name**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Conditions** | **Valid Partitions** | **Invalid Partitions** | **Valid Boundaries** | **Invalid Boundaries** |
| Customer Name | 2 to 64 chars valid Chars | < 2 Chars  > 64 Chars  Invalid Chars | 2 Chars  64 Chars | 1 Char  65 Chars  0 Chars |

**Account Number**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Conditions** | **Valid Partitions** | **Invalid Partitions** | **Valid Boundaries** | **Invalid Boundaries** |
| Account Number | 6 digits  1st non-zero | < 6 digits  > 6 digits  1st digit = 0  Non-digit | 100000  999999 | 5 digits  7 digits  0 digits |

**Loan Amount**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Conditions** | **Valid Partitions** | **Invalid Partitions** | **Valid Boundaries** | **Invalid Boundaries** |
| Loan Amount | 500-9000 | < 500  > 9000  0  Non-numeric | 500  9000 | 499  9001 |

Decision Table:🡪

* Decision Table:
  + A table showing combinations of inputs and outputs which can be used to design the test cases.
  + This technique is also referred as *cause-effect* table.

Decision Table for Login Test:🡪

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Rule 1 | Rule 2 | Rule 3 |  |
| **Conditions** | | | | |
| Valid User Name | **False** | **True** | **True** |  |
| Valid Password | **-** | **False** | **True** |  |
| **Actions** | | | | |
| Login accepted | **False** | **False** | **True** |  |