**Version History**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Ver. No.** | **Authors** | **Date** | **Reviewers** | **Review Date** | **Release Date** |
| 1.0 | Application Development Team | 27-Aug-2018 | QMF | 31-Aug-2018 | 03-Sep-2018 |
| 2.0 | Application Development Team | 10-Dec-2019 | QMF | 13-Dec-2019 | 16-Dec-2019 |
| 3.0 | Application Development Team | 02-Nov-2020 | QMF | 06-Nov-2020 | 10-Nov-2020 |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

**Change History**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Ver. No.** | **Section** | **Date** | **Change Information** | **RFC No.** |
| 1.0 | All | 03-Sep-2018 | New Release | - |
| 2.0 | All | 16-Dec-2019 | New Release | - |
| 3.0 | All | 10-Nov-2020 | Annual Review | - |
|  |  |  |  |  |
|  |  |  |  |  |

**GUIDELINES FOR TESTING**

1. **Objectives**

The objective of this document is to assist in formulating the Testing Strategy.

1. **Scope**

This document is applicable to all the projects that deliver software engineering services.

1. **Document Structure, Glossary of Terms, etc.**

* Definition
* Testing Techniques

1. **Definition**

| **Testing Type** | **Explanation / Description** |
| --- | --- |
| **Acceptance Criteria** | The criteria that a system or component must satisfy in order to be accepted by a user, customer |
| **Acceptance Testing** | Formal testing conducted to determine whether or not a system satisfies its acceptance criteria to enable the customer to determine whether to accept the system. |
| Black Box Testing | A system or component, whose inputs, outputs, and general function are known but the contents or implementation are unknown |
| **Boundary Value** | A data value that corresponds to a minimum or maximum input internal or output value specified for a system or component. |
| **Branch Testing** | Testing designed to execute each decision point in a program. |
| **Defect** | Non-fulfillment of an intended usage requirement or reasonable expectation, including one concerned with safety.   * **Producers viewpoint** **:** a product requirement that has not been met * **Customer’s viewpoint :** anything that is not present in the statement of requirements which define the product causes customer dissatisfaction. |
| **Integration**  **Testing** | Combine testing of various modules. |
| **Program Specification** | A detailed design document specifying how the program unit is implemented. This document often contains pseudo code. |
| **Regression**  **Testing** | Testing of all the test cases every time new build will get released. |
| **System testing** | The testing conducted on a complete, integrated system to verify that the system's compliance with its specified requirements |
| **Test** | An activity in which a system or component is executed under specified conditions, the results are observed and recorded |
| **Test approach** | Identification of test cases using appropriate methodologies, which ensures that all software features identified are tested under specified conditions by comparing actual behavior with the requirement specification |
| **Test case** | Documentation specifying the sequence of actions for the execution of the test. It specifies the setup, the inputs and expected results for a particular test |
| **Test coverage** | The degree to which a given test or set of tests addresses all specified requirements for a given system |
| **Test criteria** | The criteria that a system must meet in order to pass a given test. |
| **Test Cycle** | An execution of test cases as specified in the test plan. A successful cycle ends with meeting of test exit criteria an unsuccessful cycle requires execution of another cycle. |
| Test Data | A subset of the domain used in testing processing |
| **Test design** | Documentation specifying the details of a test approach for a feature or a combination of software features and identifying associated tests. |
| **Test driver** | A software module used to invoke a module under test and, often, provide test inputs, control and monitor execution, and report test results |
| **Test item** | A software item, which is an object of testing. |
| **Test Phase** | A phase in which test is executed. |
| Test Plan | A document describing the scope, approach, resources, and schedule of intended test activities. It identifies test items, the features to be tested, the testing tasks, etc. |
| **Test stub** | A skeletal or special purpose implementation of a software unit used to develop or test a unit that calls or is otherwise dependent on it. |
| **Test summary form** | A form summarizing the execution of the tests and its result. |
| **Testability** | The degree to which a system or component facilitates the establishment of test criteria and the performance of test to determine whether those criteria have been met. |
| Testing | Examination of behavior of the program on sample data |
| **Unit Testing** | Testing of source code by an individual for defects |
| **White Box testing** | A system or component, whose inputs, outputs, and general function are known |
| **Alpha Testing** | Alpha tests involve running and operating the software in production mode for a pre-specified period. It is usually performed by end users inside the developing company but outside the development organization. |
| **Beta Testing** | Beta tests involve running and operating the software in production mode for a pre-specified period. It is usually performed by a selected subset of actual customers outside the company, before the software is made available to all customers |

1. **Testing Techniques**

## Flow / Coverage Based

* + 1. **Statement Coverage**

100% statement coverage means that test data will ensure that every statement is executed at lease once during testing

* + 1. **Branch Coverage**

100% branch coverage means that test data will ensure that every branch is executed at lease once during testing

* + 1. **Condition Coverage**

100% condition coverage means that test data will ensure that every unitary condition is executed at least once during testing

* + 1. **Multiple Condition Coverage**

100% multiple condition coverage means that test data will ensure that every combination of unitary conditions (in compound conditions) is executed at least once during testing

* + 1. **Full Path Coverage**

100% path coverage means that test data will ensure that every path is executed at least once during testing

## Domain based

Here the testing techniques look at inputs and outputs and derive test cases based on the analysis of the input and output domains.

* + 1. **Equivalence Partitioning**

A systematic process which identifies a set of **classes** of input condition to be tested, where each class is representative of a large set of other possible tests, and minimizes the number of test cases required to cover those conditions. This involves two distinct steps:

1. **Identifying the Equivalence Classes (EC)**

For each external input:

If the input specifies a range of valid values, define one valid EC (within the range) and two invalid EC (one outside each end of the range)

E.g. If the input requires a month in the range of 1 - 12, define one valid EC for months 1 through 12 and two Invalid EC (months < 1 and month > 12)

If the input specifies the number (N) of valid values, define one valid EC and two invalid EC (none, and more than N).

E.g. If the input requires the titles of at least 3 but no more than 8 books, then define one valid EC and two invalid EC (< 3 and > 8 books).

If the input specifies a set of valid values, define one valid EC (within the set) and one invalid EC (outside the set).

E.g. If the input requires one of the names TOM, DICK or HARRY, then define one valid EC (using one of the valid names) and one invalid EC (using the name JOE).

If there is reason to believe that the program handles each valid input differently, then define one valid EC per valid input.

If the input specifies a must **be** situation, define one valid EC and one invalid EC.

E.g. If the first character of the input must be numeric, then define one valid EC where the first character is a number and one invalid EC where the first character is not a number

If there is reason to believe that elements in an EC are not handled in an identical manner by the program, subdivide the EC into smaller ECs.

1. **Identifying the Test Cases**

Assign a unique number to each EC

Until all valid EC have been covered by test cases, write a new test case covering as many of the uncovered EC as possible

Until all invalid EC have been covered by test cases, write a test case that covers one, and only one, of the uncovered invalid EC

If multiple invalid EC are tested in the same test case, some of those tests may never be executed because the first test may mask other tests or terminate execution of the test case.

* + 1. **Boundary Value Analysis**

A variant (refinement) of equivalence partitioning, with two major differences:

Rather than selecting any element in an EC as being representative, elements are selected such that each edge of the EC is the subject of a test.

Rather than focusing exclusively on input conditions, output conditions are also explored by defining output EC.

Valid input test cases:

1. One normal value between two ends
2. Two end value itself
3. Two near to each ends values

Invalid input test cases:

1. Two values just beyond ends

E.g. If the input requires a real number in the range [0, 90] degrees, then write test cases for 38, 0, 90, 3, 87 and -0.001, 90.001.

E.g. If the input requires the titles of at least 3 but no more than 8 books, then write test cases for 5, 3, 8, 4, 7 and 1, 9 books.

* + 1. **Error Guessing**

An ad-hoc approach, guided by intuition and experience, to identify tests which are presumed likely to expose errors. The basic idea is to make a list of possible errors or error-prone situations and then develop test based on the list.

Some common things to try:

1. Empty or null lists / strings
2. Zero instances / occurrences of an item
3. Blanks or null characters in strings
4. Negative numbers

## Population Analysis

Used to identify kinds and frequency of data in production environment. It uses existing production data either in the form of production files, tables, manual files or files from other similar systems. It could give the tester things not clear in / different from / additional to specifications codes / values used in production, unusual data conditions, type, frequency of transactions, types of incorrect transactions.

## Monkey Testing

Testing done with the intention of breaking the system. Should be done as a part of the System testing Phase.