**Test Plan**



In conjunction with



For Team 15

# 

# 

# 

# **ABSTRACT**

A test plan is a detailed document that outlines the test strategy, [Testing](https://www.guru99.com/software-testing.html) objectives, resources (manpower, software, hardware) required for testing and test schedule. Testing plan dependence on requirements documentation, architecture documentation and design documentation of the project.

# 

# TABLE OF CONTENTS

[**ABSTRACT**](#_odfdvmtu7qxt) **1**

[**TABLE OF CONTENTS**](#_d2qm5lqz07y) **2**

[**INTRODUCTION**](#_yml710tnugzw) **3**

[**REQUIREMENTS/SPECIFICATIONS-BASED SYSTEM LEVEL TEST CASES**](#_7udgjbj3cqtc) **3**

[Specification-Based techniques](#_evwsy4ij3hvm) 3

[Use Case Testing](#_vc4508sg3rjd) 3

[Decision Table Testing](#_meaojpk00is1) 3

[Equivalence Partitioning](#_xtru3lqabh86) 3

[Boundary Value Analysis](#_uy66wi2ga4zm) 4

[State Transition Diagrams](#_criu7ltihk9b) 4

[**TRACEABILITY OF TEST CASES TO USE CASES**](#_trwst88jp23v) **4**

[**TECHNIQUES FOR TEST GENERATION**](#_m2lg9j7thd79) **4**

[Structure-Based techniques](#_j3jvt9ba9q7d) 4

[Statement Testing & Coverage](#_uygsgqjqvrk8) 4

[Decision Testing Coverage](#_fq6hu0g9cpmt) 4

[Condition Testing](#_7r6oatcx1g91) 5

[Multiple Condition Testing](#_leutjn22pk79) 5

[All Path Testing](#_feytcoc7gfo7) 5

[Test coverage](#_3wycavfsdh05) 6

[**EVIDENCE THE TEST CASES, DOCUMENT HAVE BEEN PLACED UNDER CONFIGURATION MANAGEMENT**](#_3mkoa72qpair) **6**

[**REFERENCES**](#_4rftaoe8b6zw) **7**

# INTRODUCTION

* This test plan is a detailed document that outlines the test strategy, [Testing](https://www.guru99.com/software-testing.html) objectives, resources (manpower, software, hardware) required for testing and test schedule. Testing plan dependence on requirements documentation, architecture documentation and design documentation of the project.
* Choosing the right test case design technique is important in testing plan process. The main purpose of test case design techniques is to test the functionalities and features of the software with the help of effective test cases. Therefore we are going to use specification-based techniques and structure-based techniques.

.

# REQUIREMENTS/SPECIFICATIONS-BASED SYSTEM LEVEL TEST CASES

## Specification-Based techniques

Uses the specification of the program as the point of reference for test data selection and adequacy. A specification can be any thing like a written document, collection of use cases, a set of models or a prototype. There are 5 types of specification-based techniques that we are going to use.

## Use Case Testing

Use case testing involves the use of use cases. In this technique, the application is tested using use-cases that represent the interaction of the application under test with its different users or actors.

## Decision Table Testing

In this technique, test cases are designed on the basis of the [decision tables](https://reqtest.com/requirements-blog/a-guide-to-using-decision-tables/) that are formulated using different combinations of inputs and their corresponding outputs based on various conditions and scenarios adhering to different business rules.

## Equivalence Partitioning

Testing cannot be done with all possible input conditions for validating any UI element, feature. We can divide the range in to partitions, take one from each possible partition and test that particular text box. Equivalence Partitions are also known as equivalence classes. It can be used to achieve input and output coverage goals. It can be applied to human input, input via interfaces to a system, or interface parameters in integration testing.

## Boundary Value Analysis

Boundary value analysis testing involves creating test data based on the boundary values of the equivalence classes. E.g. equivalence classes for test data with value lying between 0 to 100, the boundary values would be 0 and 100.

## State Transition Diagrams

In this technique, the software under test is perceived as a system having a finite number of states of different types. The transition from one state to another is guided by a set of rules. The rules define the response to different inputs. This technique can be implemented on the systems which have certain workflows within them.

# TRACEABILITY OF TEST CASES TO USE CASE

|  |  |
| --- | --- |
| **Test case** | **Use case** |
| Does execute button display expected outcome within a 3 seconds time frame | Display GUI |
| Is output formatted in JSON as expected | Output JSON Formatted File |
| Run multiple vulnerability scans and test the consistency of system vulnerabilities | Retrieve System Vulnerabilities for OS Version |
| Test that operating version requests are in fact pulling the correct operating system with known operating systems on the network | Request Operating System Versions |
| Check that packet payload parsing is correct and returning consistent outputs that correlate with original packet | Parse Network Packet Payloads |
| Test that network packets are being collected from expected IP’s on the network | Capture Network Packets |

# TECHNIQUES FOR TEST GENERATION

We are going to use Structure-Based techniques for test generation.

## Structure-Based techniques

The structure-based technique is based on the internal structure of the software. This technique exhaustively tests the developed code. Developers who have complete information of the software code, its internal structure, and design help to design the test cases. This technique is further divided into five categories. Structural testing is more concerned with how system does it rather than the functionality of the system. It provides more coverage to the testing. Its helps in performing a thorough testing on software.

## Statement Testing & Coverage

Statement coverage is a technique where all the statements at the source code are executed at least once. To calculate Statement Coverage, find out the shortest number of paths following which all the nodes will be covered.

## Decision Testing Coverage

This technique is also known as branch coverage is a testing method in which each one of the possible branches from each decision point is executed at least once to ensure all reachable code is executed. This helps to validate all the branches in the code. This helps to ensure that no branch leads to unexpected behavior of the application.

## Condition Testing

Testing the condition outcomes(TRUE or FALSE). So, getting 100% condition coverage required exercising each condition for both TRUE and FALSE results using test scripts(For n conditions we will have 2n test scripts).

## Multiple Condition Testing

In the MCC coverage metric, all statements must be executed and all combinations of truth values in each decision must occur at least once to reach full coverage. The coverage of a program is the number of executed statement blocks and condition combinations divided by their total number in the program.

## All Path Testing

Testing the independent paths in the system(paths are executable statements from entry to exit points).

|  |  |
| --- | --- |
| **Black box testing** | **White box testing** |
| Testing is broadly based on software requirements and specifications. Black Box Testing is a technique in which tester is unaware about the internal structure or code of the software. | White Box Testing is also known as open, transparent or glass box testing. In white box testing, the tester has prior knowledge of the code and accordingly prepares the test case. |
| No knowledge of programming is required. | It is mandatory to have knowledge of programming. |
| It is least time consuming. | It is most time consuming. |
| It is mostly done by software testers. | It is mostly done by software developers. |
| Can be done by trial and error ways and methods. | Data domains along with inner or internal boundaries can be better tested. |
| It is functional test of the software. | It is a structural test of the software. |

Testing a software system is a continuous effort that can be hard to get right. Our team will test things at various levels, often repeating business logic, causing an overhead of tests to support. We are going to use the most popular way of measuring software test quality, which is test coverage.

## Test coverage

In order to understand if we have written enough tests for our codebase, a good metric to measure is code coverage. If we have tested 100% of our codebase, then we should have no regression issues. However, 100% should not be the target metric value. 100% test coverage is near impossible to attain. Not only it is extremely hard to achieve, it can also be a complete waste of time. When testing a system, there always needs to be a trade-off between time and value. If our goal is solely to reach 100% code coverage, then there might be many hours spent giving little value to the end-product.

# EVIDENCE THE TEST CASES, DOCUMENT HAVE BEEN PLACED UNDER CONFIGURATION MANAGEMENT

<https://github.com/Narthexes/matilda-senior-design-project/blob/master/Project%20Plan.docx>

# REFERENCES

N/A