iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 3 **Issue:** III **Month of publication:** March 2015

DOI:

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)

Testing Techniques for Test Cases Generation

Prof. A. A. Shaikh[1], Prof. P.P. Gaddekar[2]
[1,2]Department of Computer Technology, P. Dr. V. Vikhe Patil Institute of Technology and Engineering (Polytechnic), Pravaranagar, Ahmednagar, Maharashtra

Abstract— Software testing is the process of executing a program or application with the intent of finding software bugs (errors or other defects). Testing cannot establish that a product functions properly under all conditions but can only establish that it does not function properly under specific conditions. It also verifies and validate whether the program is working correctly with no bugs no not. There are different levels of testing- Unit, Integration, Component interface, System and Acceptance. Unit testing referred to as testing in small whereas Integration, Component interface and System testing are referred to as testing in large. At last the system is delivered to the user for Acceptance testing. Various testing techniques available for designing of test cases. This paper basically deals with various techniques available to design software testing test cases from the structure of the system.

Keywords— Software testing, Unit, Integration, Component interface, System Testing, Acceptance Testing, Statement overage, Brach Coverage, Path Coverage, Testing Sequence

I. INTRODUCTION

Testing is the process of evaluating a system or its component(s) with the intent to find whether it satisfies the specified requirements or not. In simple words, testing is executing a system in order to identify any gaps, errors, or missing requirements in contrary to the actual requirements. According to ANSI/IEEE 1059 standard, Testing can be defined as - A process of analyzing a software item to detect the differences between existing and required conditions (that is defects/errors/bugs) and to evaluate the features of the software item.Software testing is the process of executing a software system to determine whether it matches its specifications and executes in its intended environments. A software failure occurs when a piece of software does not perform as required and as expected. In software testing, the software is executed with input data or test cases and output of the
software is observed. The testing process can be divided into three steps- test case generation, test case execution and test evaluation. A test case is the triplet [I, S, O], where I is the input given to the system and S is the state of the system at which input is given and O is the expected output of the system.

A. Levels Of Testing

There are different levels during the process of testing. Levels of testing include different methodologies that can be used while conducting software testing. Testing starts from testing a simple screen, sub-modules, module, combination of modules, and then finally testing the complete system.

1) *Unit Testing:* Software product is made up of many units, each unit needed to be tested to find whether they have implemented the design correctly or not. It may involve designing and usage of stub/driver to execute units as they may not be executed alone. It is a level of the software testing process where individual units / components of a software/ system are tested. The purpose is to validate that each unit of the software performs as designed.
2) *Integration Testing:* It is a level of the software testing process where individual units are combined and tested as a group. The purpose of this level of testing is to expose faults in the interaction between integrated units. It is both a type of testing and a phase of testing. It is a set of interactions all defined interactions among components needed to be tested. (How the system or modules work together) It involves testing of many units by combining them together to form a sub module. It is done by developers, If its executed independently the done by testers. It ensures that the units tested independently are also going to work together correctly. It is mainly refers to detail or low-level design.
   Integration testing is sub-divided as follows:
   a) *Big-bang approach.*
   b) *Top-down approach*
   c) *Bottom-up approach*
   d) *Bi-directional integration apporach.*
3) *Component Interface Testing:* The practice of component interface testing can be used to check the handling of data passed between various units, or subsystem components, beyond full integration testing between those units. One option for

interface testing is to keep a separate log file of data items being passed, often with a timestamp logged to allow analysis of thousands of cases of data passed between units for days or weeks. Tests can include checking the handling of some extreme data values while other interface variables are passed as normal values.

4) *System Testing:* System testing tests the system as a whole. Once all the components are integrated, the application as a whole is tested rigorously to see that it meets the specified Quality Standards. This type of testing is performed by a specialized testing team.
5) *Acceptance Testing:* This is arguably the most important type of testing, as it is conducted by the Quality Assurance Team who will gauge whether the application meets the intended specifications and satisfies the client's requirement. The QA team will have a set of pre-written scenarios and test cases that will be used to test the application.
6) *Alpha Testing:* Alpha testing is the system testing performed by the development team..
7) *Beta Testing:* Beta testing is the system testing performed by the customer himself after the product delivery to determine whether to accept the delivered product or to reject it.

II. SOFTWARE TESTING TECHNIQUES

The purpose of software testing is to identify all the defects in a program. There are many techniques available for designing of test cases from the structure of the system as given below.

A. Statement Coverage

Statement Coverage is a code coverage metric that tells you whether the flow of control reached every executable statement of source code at least once. It identifies which statements in a method or class have been executed. The benefit is its ability to identify which blocks of the code have not been executed

B. Branch Coverage

Branch Coverage is also called as Decision Coverage. Whenever there are two or more possible exits from the statement like an IF statement, a DO-WHILE or a CASE statement it is known as decision because in all these statements there are two outcomes, either TRUE or FALSE. With the loop control statement like DO-WHILE or IF statement the outcome is either TRUE or FALSE and decision coverage ensures that each outcome (i.e TRUE and FALSE) of control statement has been executed at least once. Alternatively you can say that control statement IF has been evaluated both to TRUE and FALSE. The formula to calculate decision coverage is: **Decision Coverage**=(Number of decision outcomes executed/Total number of decision outcomes)*100% Researches in the industries have shown that even if through functional testing has been done it only achieves 40% to 60% decision coverage. Decision coverage is stronger that statement coverage and it requires more test cases to achieve 100% decision coverage.

C. Path Coverage

In this the test case is executed in such a way that every path is executed at least once. All possible control paths taken, including all loop paths taken zero, once, and multiple (ideally, maximum) items in path coverage technique, the test cases are prepared based on the logical complexity measure of a procedural design. In this type of testing every statement in the program is guaranteed to be executed at least one time. Flow Graph, Cyclomatic Complexity and Graph Metrics are used to arrive at basis path

Cyclomatic Complexity

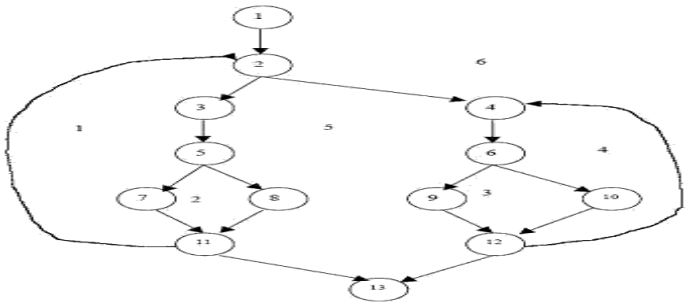
Cyclomatic Complexity for a flow graph is computed in one of three ways:

The numbers of regions of the flow graph correspond to the Cyclomatic complexity.

Cyclomatic complexity, V(G), for a flow graph G is defined as
$V(G) = E - N + 2$
where E is the number of flow graph edges and N is the number of flow graph nodes.

Cyclomatic complexity, V(G), for a graph flow G is also defined as
$V(G) = P + 1$
Where P is the number of predicate nodes contained in the flow graph G.

Example : Consider following flow graph



Region, R= 6
Number of Nodes = 13
Number of edges = 17
Number of Predicate Nodes = 5
Cyclomatic Complexity, V(C) :
V(C) = R = 6;
Or
V(C) = Predicate Nodes + 1
=5+1 =6
Or
V(C)= E-N+2
= 17-13+2

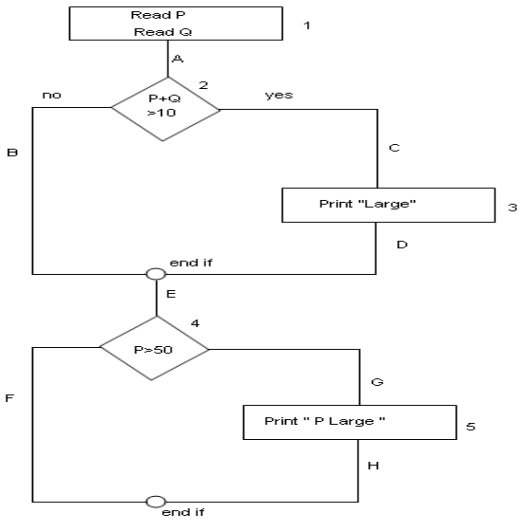
Nodes represent entries, exits, decisions and each statement of code. Edges represent non-branching and branching links between nodes.

How to calculate Statement Coverage , Branch Coverage and Path Coverage :

Example:

```
Read P
Read Q
IF    P+Q > 100 THEN
          Print “Large”
ENDIF
If P > 50 THEN
          Print “P Large”
ENDIF
```

Flowchart-



Statement Coverage (SC):

To calculate Statement Coverage, find out the shortest number of paths following which all the nodes will be covered. Here by traversing through path 1A-2C-3D-E-4G-5H all the nodes are covered. So by traveling through only one path all the nodes 12345 are cove red, so the Statement coverage in this case is 1.

Branch Coverage (BC):

To calculate Branch Coverage, find out the minimum number of paths which will ensure covering of all the edges. In this case there is no single path which will ensure coverage of all the edges at one go. By following paths 1A-2C-3D-E-4G-5H, maximum numbers of edges (A, C, D, E, G and H) are covered but edges B and F are left. To covers these edges we can follow 1A-2B-E-4F. By the combining the above two paths we can ensure of traveling through all the paths. Hence Branch Coverage is 2. The aim is to cover all possible true/false decisions.

Path Coverage (PC):

Path Coverage ensures covering of all the paths from start to end.

All possible paths are-

1A-2B-E-4F
1A-2B-E-4G-5H
1A-2C-3D-E-4G-5H
1A-2C-3D-E-4F

So path coverage is 4.

Thus for the above example SC=1, BC=2 and PC=4.

Memorize these….

100% LCSAJ coverage will imply 100% Branch/Decision coverage
100% Path coverage will imply 100% Statement coverage
100% Branch/Decision coverage will imply 100% Statement coverage
100% Path coverage will imply 100% Branch/Decision coverage
Branch coverage and Decision coverage are same.

*LCSAJ = Linear Code Sequence and Jump

III. TESTING SEQUENCE

A tester is the nearest person to the real-time end user of the product who will be using the product. He acts like virtual end user who will use the product in real-time scenarios. Tester needs to understand not just his/her features, but also other features in the product which interact with his/her features, thus a tester knows whole of the product under testing. The testing sequence is given below:

A. Functionality

This is the first thing to test in a feature under test because this is what the users are paying for. For example: - Test if you can save a file on a specific location and verify the file"s existence on that location.

B. GUI

Verify all the fields on the graphical user interface for data validation of those fields. In early stage of product testing, these errors are very much existent and you can easily find them.

C. Error Conditions

Test the software to verify if it handles the errors gracefully and shows the user-friendly messages to the end user.

D. Stress testing

Test the software by subjecting it to stressed conditions. Usually testers tend to test in non-stressed conditions. For Examples:- Try to save a huge file on low memory disk, or try to print a document with 2500 page in it, try to save a form with maximum data in all the fields etc.

E. User Scenarios

User Scenarios requires feature to work. User scenarios can be executed earlier after functionality testing to see if the feature

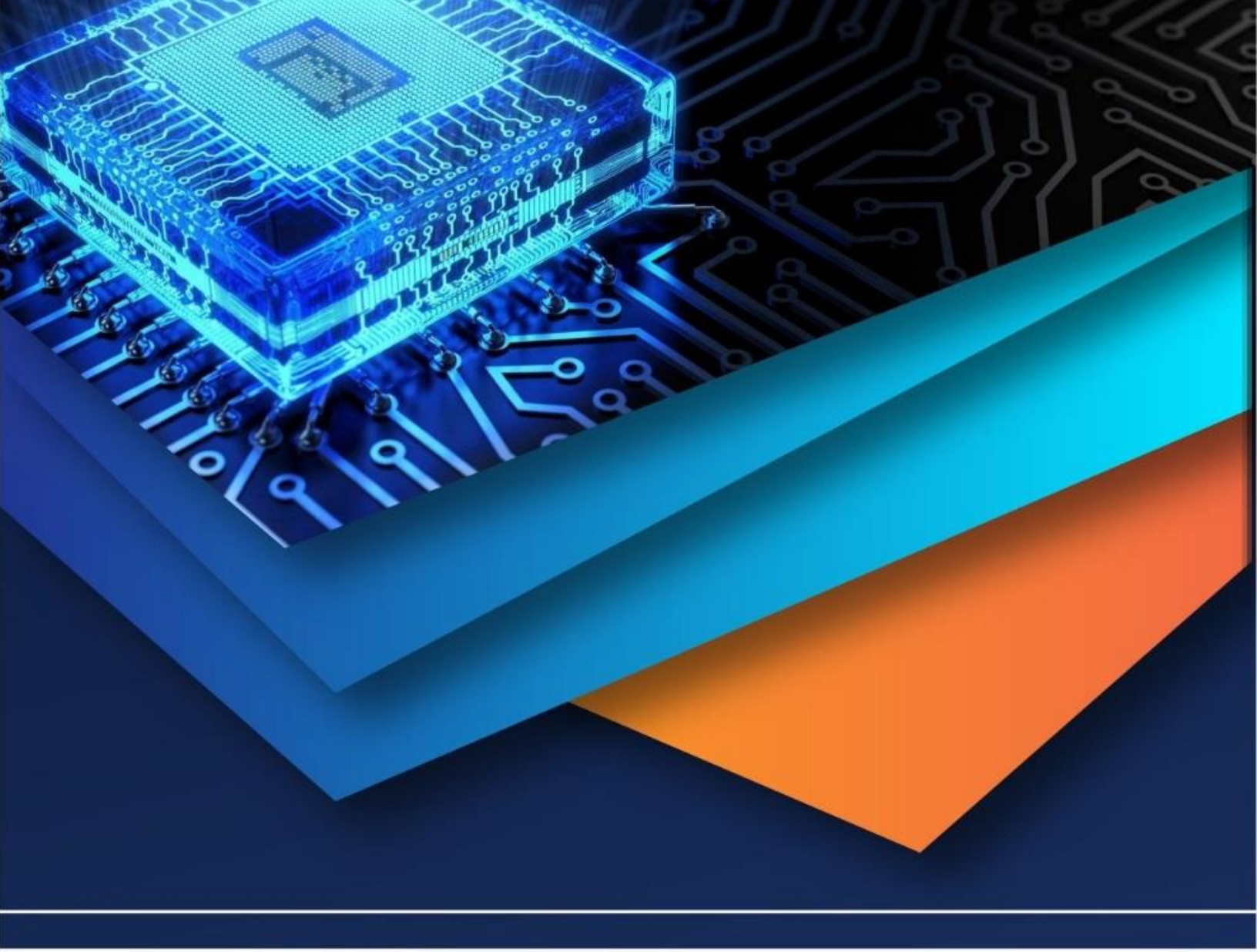
interacts well with other product features and generates correct results as expected.

IV. CONCLUSIONS

Software testing is very important element of software development life cycle (SDLC). For a software product, usually around 50% of money and resources go under software testing. Software testing can furnish excellent results if done properly and effectively. Test cases are most important elements of software testing. If we design test cases in better manner and in properly way using test cases designing techniques then we can result with better software testing of a software product. By doing this we will get the result as we want in the requirement specified in the software requirement specification (SRS). This paper mainly deals with testing techniques available for designing of test cases. In the future, we can implement these test cases design techniques for real-time scenarios projects.

REFERENCES

[1] Antonia Bertolina, "*Software Testing Research and Practice*", Proceedings of the Abstract state machines 10th international conference on advances in theory and practice.
[2] Mohd. Ehmer Khan, "*Different forms of software testing techniques for finding errors*", IJCSI International Journal Of Computer Science Issues, Vol. 7, Issue 3, No. 1, May 2010.
[3] Sahil Batra, Dr. Rahul Rishi, "*Improving Quality Using testing strategies*", Journal of Global Research In Computer Science, Volume 2, No. 6, June 2011.
[4] Abhijit A. Sawant, Pranit H. Bari and P.M. Chawan, "*Software testing techniques and Strategies*", International Journal of Engineering Research and Applications (IJERA), pp.980-986, Vol. 2, Issue 3, May-June 2012.
[5] Dolores R. Wallace, Laura M. Ippolito, Barbara B. Cuthill, "*Reference Information for the software Verification and Validation Process*", DIANE Publishing, 1996.
[6] Drake, T.(1996), "*Measuring Software quality: A case study*". IEEE Computer, 29(11), 78-87.
[7] Sheetal Thakre, Savita Chavan, Prof. P.M. Chawan, "*Software Testing Strategies and Techniques*", International Journal of Emerging Technology and Advanced Engineering, pp.567-569, Vol. 2, Issue 4, April 2012.
[8] Edward L. Jones "Grading student programs- a software testing", Proceedings of the fourteenth annual Consortium for computing Sciences in Colleges, 2000.
[9] Ian Somerville, "*Software Engineering*", Addison-Wesley, 2001[10] Rajib Mall, "*Fundamentals of software engineering*", The prentice hall of India, 3rd Edition, 2011. [11] Peter Sestoft, "*Systematic software testing*", Version 2, 008-02-25.
[12] Gaurav Saini, Kestina Rai, "*An Analysis on Objectives, Importance and Types of Software Testing*", International Journal of Computer Science and Mobile Computing (IJCSMC), Vol. 2, Issue 9, September 2013, pp.18-23. [13] Rajat kumar Bal, "*Software Testing*".



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY