



Prioritization and Path Selection Criteria: A Review

Suruchi*, Arashdeep Kaur
CSE, Punjab Technical University
Punjab, India

Abstract— *One of the major aspect to analyze a software system under the testing process is the path testing. The path testing can be defined either a white box or the black box testing approach. In this paper, we have defined basic testing and software module analysis based on the path selection criteria. We can choose the best path using the path selection criteria which is further helpful for path testing. We can also prioritize the test cases for path testing by using certain parameters. Using which, the best optimal path is selected.*

Keywords: *Testing, Path selection, Prioritization, Parameters.*

I. INTRODUCTION

Software development is not a precise science, and humans being as error prone they are, software development must be accompanied by quality assurance activities. It is typical for developers to spend around 40% of the total project time on testing. For life critical software (e.g. flight control, reactor monitoring), testing can cost 3 to 5 times as much as all other activities combined. The destructive nature of testing requires that the developer discard preconceived notions of the correctness of his/her developed software. This means that testing must be done from an entirely different perspective from that of a developer. Testing must be done by considering the end user in mind, that how he would be using this software...

In software development project, errors can be come in at any stage during development. The main causes of errors are

1. Not obtaining the right requirements,
2. Not getting the requirements right, and
3. Not translating the requirements in a clear and understandable manner so that programmers implement them properly.

There are techniques available for detecting and eliminating errors that originate in various stages. However, no technique is perfect. Testing is basically a process to detect errors in the software product. It is a very important phase during the software development. Before going into the details of testing techniques one should know what errors are. In day-to-day life we say whenever something goes wrong there is an error. This definition is quite vast. When we apply this concept to software products then we say whenever there is difference between what is expected out of software and what is being achieved, there is an error.

For the output of the system, if it differs from what was required, it is due to an error. This output can be some numeric or alphabetic value, some formatted report, or some specific behavior from the system. In case of an error there may be change in the format of out, some unexpected behavior from system, or some value different from the expected is obtained. These errors can due to wrong analysis, wrong design, or some fault on developer's part.

All these errors need to be discovered before the system is implemented at the customer's site. Because having a system that does not perform as desired be of no use. All the effort put in to build it goes waste. So testing is done. And it is equally important and crucial as any other stage of system development. For different types of errors there are different types of testing techniques. In the section that follows we'll try to understand those techniques.

II. OBJECTIVES OF TESTING

First of all the objective of the testing should be clear. We can define testing as a process of executing a program with the aim of finding errors. To perform testing, test cases are designed. A test case is a particular made up artificial situation upon which a program is exposed so as to find errors. So a good test case is one that finds undiscovered errors. If testing is done properly, it uncovers errors and after fixing those errors we have software that is being developed according to specifications.

A. Test Information Flow

Testing is a complete process. For testing we need two types of inputs. First is software configuration. It includes software requirement specification, design specifications and source code of program. Second is test configuration. It is basically test plan and procedure.

Software configuration is required so that the testers know what is to be expected and tested whereas test configuration is testing plan that is, the way how the testing will be conducted on the system. It specifies the test cases and their expected value. It also specifies if any tools for testing are to be used. Test cases are required to know what specific situations need to be tested. When tests are evaluated, test results are compared with actual results and if there is some error, then debugging is done to correct the error. Testing is a way to know about quality and reliability of the software. Then the error rate that is the occurrence of number of errors is evaluated. This data can be used to predict the occurrence of errors in future.

B. White Box Testing

White box testing uses the internal mechanism of a system to create test cases, which uses calculated paths to discover unidentified bugs within the system. This type of testing is trying to enforce the quality of the software system however “white box testing” is a cost effective method and is compared very closely to “Black box testing”. The main jobs of these two functions have the same purpose however it is majorly debated which one is more efficient and effective. Black box testing concentrates mainly on the outputs of the system to identify bugs however this function waits till a malfunctioning error has occurred. As explained above White box testing is used to ensure that the code is complete and to the correct standard of the software mechanism. Statistics have proven that by using a complete and precise systematic test design, will ensure that the majority of bugs within the system will be identified.

When looking at white box testing in more detail it involves checking and ensuring that every program statement is error free. White box testing allows:

- ❖ Data Processing
- ❖ Calculation correctness tests
- ❖ Software Qualification tests
- ❖ Maintainability tests
- ❖ Reusability tests

➤ Data Processing and Calculation Correctness tests

Within the main two concepts of Calculation Correctness tests and Data processing, when a test case has been created (path) it is then proceeded to be tested and verified which will ensure the correctness of the software program. This procedure is done in sequence for every test case created, this gives the user the opportunity to test whether the code / program has been implemented correctly to the requirements / specification.

➤ Software Qualification tests

When going through the software qualification tests it looks directly to the software code itself. This tests works with the “coding standards” and work instructions set by the client or developer.

➤ Maintainability tests

The maintainability tests install special features to detect failures, module structures that support software development, extensions and improvements of the software system. The reusability tests are part of an example of this, as explained below:

➤ Reusability tests

Reusability tests are performed on the package which examines all of the reused software that had been implemented into the software package. Addition to this function it also adapts to the program allowing the present code to be reusable for future preferences.

When looking at the White box testing in more detail and analysing its procedures it becomes aware that when testing each test case / path maybe become unrealistic and unfeasible. We have to take into account on the amount of coverage of all the possible paths and the amount of code that is produced. There are two possible solutions for this situation:

- ❖ Line Coverage – this is to test and cover over all of the program code and the coverage is calculated by the percentage of lines covered.
- ❖ Path Coverage – this is to test and cover all the available paths in the system and the coverage is calculated by the percentage of paths covered.

C. Correctness tests and Path coverage

Path testing’s main goal is to go through the system covering all possible paths, this is tested with all condition such as IF-THEN-ELSE and DO WHILE statements. To take white box into account it is very impractical due to the amount of resources needed for this type of testing. Path testing is used from the start of a method to when it finishes, each path shows the flow of execution. By looking at white box complete path coverage will show how extreme this type of testing is:

D. Correctness Tests and Line Coverage

The line coverage requires a different process to path coverage as explained above. There are different types of line coverage depending on the user’s requirements and specification of the software system. One function will be to run a full complete line coverage process which will consist of each line of code being compiled and executed at least once. This will be shown as a percentage of the lines compiled during the testing process. Line coverage may also perform a less efficient method test which consists of testing fewer test cases which will make the system more vulnerable to bugs and system errors.

III. BASIS PATH TESTING

In the 1970's, Thomas McCabe came up with the idea of using a vector space to carry out path testing. A vector space is a set of elements along with certain operations that can be performed upon these elements. What makes vector spaces an attractive proposition to testers is that they contain a basis. The basis of a vector space contains a set of vectors that are independent of one another, and have a spanning property; this means that everything within the vector space can be expressed in terms of the elements within the basis. What McCabe noticed was that if a basis could be provided for a program graph, this basis could be subjected to rigorous testing; if proven to be without fault, it could be assumed that those paths expressed in terms of that basis are also correct. The method devised by McCabe to carry out basis path testing has four steps. These are:

1. Compute the program graph.
2. Calculate the cyclomatic complexity.
3. Select a basis set of paths.
4. Generate test cases for each of these paths

In order to show the workings of McCabe's basis path method, we will progress through each step before finally demonstrating how it could be utilized as a structural testing method. To begin, we need a program graph from which to construct a basis. This is a commonly used example, as it demonstrates how the basis of a graph containing a loop is computed. It should be noted that the graph is strongly connected; that is, there exists an edge from the sink node to the source node.

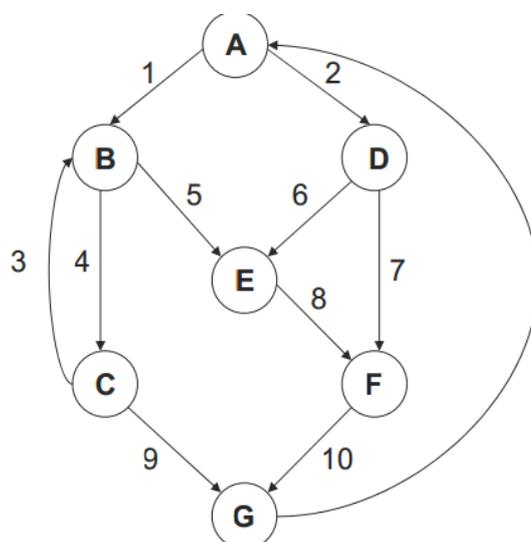


Fig 1 McCabe's strongly connected program graph

In graph theory, there exists a proof that states that the cyclomatic complexity of a strongly connected graph is the number of linearly independent circuits in a graph [Jorgensen, 2002]. McCabe realized that path testing commonly makes use of program graphs, and so felt that this theorem could be used to find a basis that could be tested. The cyclomatic complexity of a strongly connected graph is provided by:

$$\text{The formula } V(G) = e - n + p.$$

The number of edges is represented by e, the number of nodes by n and the number of connected areas by p. If we apply this formula to the graph given, the number of linearly independent

$$\begin{aligned} \text{A circuit is: } V(G) &= e - n + p \\ &= 11 - 7 + 1 = 5 \end{aligned}$$

IV. PATH SELECTION CRITERIA

There are many paths between the entry and Exit points of a typical routine.

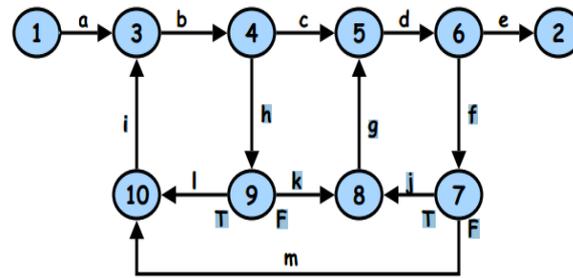
- Even a small routine can have a large Number of paths.

A. Path Testing Criteria

- ❖ We have explored 3 testing criteria from an Infinite set of strategies:
 - 1) Path Testing ():
 - 100% path coverage.
 - ❖ Execute all possible control flow paths through the program.
 - 2) Statement Testing ():
 - 100% statement coverage.
 - ❖ Execute all statements in a program at least once under some test.
 - 3) Branch Testing ():
 - 100% branch coverage.

❖ Execute enough tests to assure that every branch alternative has been exercised at least once under some test.

P1
P2
P " P " " P
!



PATHS	DECISIONS				PROCESS LINKS												
	4	6	7	9	a	b	c	d	e	f	g	h	i	j	k	l	m
<i>abcde</i>	<i>T</i>	<i>T</i>			*	*	*	*	*								
<i>abhkgde</i>	<i>F</i>	<i>T</i>		<i>F</i>	*	*		*	*	*	*				*		
<i>abhlibcde</i>	<i>TF</i>	<i>T</i>		<i>T</i>	*	*	*	*	*	*	*				*		
<i>abcdfjgde</i>	<i>T</i>	<i>TF</i>	<i>T</i>		*	*	*	*	*	*	*				*		
<i>abcdfmibcde</i>	<i>T</i>	<i>TF</i>	<i>F</i>		*	*	*	*	*	*	*				*		*

Fig 2 Example of P1 and P2 Coverage

V. EFFECTIVENESS OF PATH TESTING

- About 65% of all bugs can be caught in unit testing.
- Unit testing is dominated by path testing methods.
- Statement and branch testing dominates path testing.
- Studies show that path testing catches 50% of all bugs caught during unit testing.
- About 33% of all bugs.
- Path testing is more effective for unstructured code than for code that follows structured programming.
- Experienced programmers can bypass drawing flow graphs by doing path selection on the source.

VI. LIMITATIONS OF PATH TESTING

Path testing as a sole testing technique is limited:

1. Interface mismatches and mistakes are not caught.
2. Not all initialization mistakes are caught by path testing.
3. Specification mistakes are not caught.

VII. CONCLUSION

IN this paper we have carried out an in-depth analysis of path testing, and the different methods that exist. what we have found is that no matter which method is implemented, be it dd-path testing or the basis path testing method, the aim of path testing remains the same; to provide a test coverage metric so that the tester has information concerning the amount of faults within the code that are likely to have been uncovered. The reason behind why the test coverage metrics provided by path testing is so important is that they provide a target for testers to aim for. Without this target, when would the testing process end? It is impossible to test until all faults have been removed as this can never be guaranteed, while testing until no new faults are uncovered can be an expensive process. However, it is viable to test a program until specific test coverage metric has been achieved. This means that once a program has been fully tested in respect to a certain metric, the tester has an idea of the percentage of faults that have been uncovered.

REFERENCES

[1] [Beizer, 1978] Beizer, B., *Micro-Analysis of Computer System Performance*, 1978. New York: Van Nostrand Reinhold.

[2] BEIZER90:B. Beizer, *Software Testing Techniques, 2 Edition*, Van Nostrand Reinhold, 1990.

[3] EVANG84: M. Evangelist, "An Analysis of Control Flow Complexity", *Eighth International Computer Software and Application Conference*, pg. 388 - 396, 1984.

[4] HETZEL84: W. Hetzel, *The Complete Guide to Software Testing*, QED Information Sciences, Inc., 1984.

[5] IEEE610: ANSI/IEEE Std 610.12-1990, "Glossary of Software Engineering Terminology", The Institute of Electrical and Electronics Engineers, Inc., February, 1991.

[6] LINGER79: R. Linger, H. Mills and B. Witt, *Structured Programming: Theory and Practice*, Addison-Wesley Publishing Company, 1979.

- [7] MCCABE76: T. McCabe, "A Complexity Measure", IEEE Transactions on Software Engineering, Vol SE-2, Number 4, December 1976, pg. 308-320.
- [8] MYERS77: G. Myers, "An Extension to the Cyclomatic Measure of Program Complexity", SIGPLAN Notices, October, 1977.
- [9] NISTIR5691: NIST IR 5691, J. Lyle, D. Wallace, J. Graham, K. Gallagher, J. Poole and D. Binkley, "Unravel: A CASE Tool to Assist Evaluation of High Integrity Software", volumes 1 and 2, Department of Commerce, August, 1995.
- [10] SP500-99: NBS Special Publication 500-99, T. McCabe, "Structured Testing: A Software Testing Methodology Using the Cyclomatic Complexity Measure", December, 1982.
- [11] WEISER84: M. Weiser, "Program Slicing", IEEE Transactions on Software Engineering, 10:352-357, July 1984.