

An Experimental Study Of Fault Detection Rate Of Structure Testing

Monika*, Mrs.PuneetJai Kaur**

U.I.E.T,Panjab University, Chandigarh

*monikahsp@yahoo.com, ** puneetkaur79@yahoo.co.in

Abstract - Software testing is one of the major activities in software engineering to detect any undesired behaviour. This paper contributes a controlled experiment to show the fault detection rate of structural testing. Several structure testing methodologies are described and fault detection rate of structure testing is calculated that serve as metric to show the efficiency structure testing. We are aware that a single experiment has many limitations and, often, does not provide conclusive evidence. Hence, we consider this experiment a starting point and encourage other researchers to investigate the optimal mix of fault detection techniques.

Keywords: Fault detection rate, software testing, structure testing

I. INTRODUCTION

This paper is structured as follows: Section 2 describes the fault detection technique applied in the controlled experiment. Section 3 provides a description of the experiment. Section 4 presents the analysis and the experimental results. Section 5 concludes the paper.

Software Testing is the process of executing software and comparing the observed behaviour to the desired behaviour. The major goal of software testing is to discover errors in the software, with a secondary goal of building confidence in the proper operation of the software when testing does not discover errors. The role of software testing is not only to assure that a system does everything it is supposed to do, but also that a system does nothing more than what it is designed to do. Testing is now characterized as a broad and continuous activity throughout the development process ([1], pg.5) whose aim is the measurement and evaluation of software attributes and capabilities. A survey of current test practices in Australia shows that the most popular software testing metric is fault count [10].

1.1 Some terminologies.

There are some definitions related to terminologies used in testing, given by IEEE:

- **Error:** People make errors when they reason incorrectly to solve a problem.
- **Fault:** An error becomes a fault when it is written (included) in any of the developed software products.
- **Failure:** Failures occur when a software system doesn't behave as desired. This reveals a fault in software.
- **Defect:** The term defect is generally used to refer to any of these concepts.

Developers make errors for any number of reasons: from a conceptual error, through a misinterpretation etc. any of these errors can lead the developer to enter a fault in the software at some point. A fault can take many forms. At the very simplest these faults can be classed into three groups:

- Omission (something that should have been done has been overlooked)
- Excess (something that should not have been done, has been done)
- Commission (something has been done incorrectly)

Defect detection refers to the observation that the program behavior is wrong, i.e. the manifestation of a failure [6]. Defect is a general term that encompasses both failures and faults [7] [11]. To detect the software faults, which have been generated during the development process, two different strategies may be applied, static and dynamic strategies [9].

A. TYPES OF SOFTWARE TESTING TECHNIQUES.

Two fundamentally different categories of test techniques exist, black-box and white-box techniques.

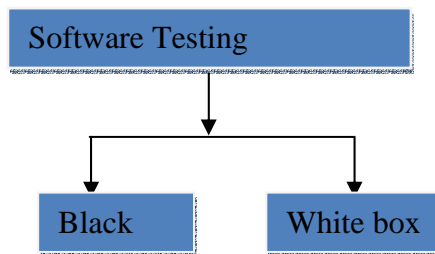


Fig 1: Types of Software testing techniques

Black-box testing: Black-box test techniques are based on requirements. They are therefore also called behavioural or functional techniques. Testing done under a black-box strategy is called black-box testing. Black-box test strategies do ignore the internal structure of the object under test.

White-box testing: White box testing strategy deals with the internal logic and structure of the code. White box testing is also called as glass, structural, open box or clear box testing. The tests written based on the white box testing strategy incorporate coverage of the code written, branches, paths, statements and internal logic of the code etc. In order to implement white box testing, the tester has to deal with the code and hence is needed to possess knowledge of coding and logic i.e. internal working of the code.

II. DEFECT DETECTION TECHNIQUE.

Structured testing forms the “white box,” or code-based, part of a comprehensive testing program. Structural tests, also known as white-box tests, are based on how a system operates[5]. The structural technique uses the source code and a program graph (G) to derive test cases. It generally establishes criteria that are predicates to be satisfied. A criterion can be used to generate test data or to evaluate the adequacy of a test set. Control flow based criteria and data flow based criteria are the best known structural criteria. Control flow based criteria require the exercising of some elements of the control flow graph: nodes, edges and paths. Data flow based criteria test definitions and uses of variables in a program . One type of testing, white box (or structural) testing, is based on covering program entities, such As statements, branches, and data-flow relationships. Some defects missed by inspection are detected by structural testing[3]. Structural testing requires a knowledge of the implementation details [4]. Structure testing aims to construct test cases that achieve 100% coverage of all branches, multiple conditions, loops, and relational operators. The coverage criteria were defined as follows [18], [20]:

- Branch coverage: Complete branch coverage requires that every branch be exercised at least once in both the true and false directions.
- Loop coverage: Complete loop coverage requires that a loop condition be executed once, several times and also should be skipped (without ever entering the loop) in some test condition.
- Multi-condition coverage: This has a stronger requirement than branch coverage. It checks for all parts of a logical expression being used. That is, each of the logical expression components must evaluate to TRUE in some test, and to FALSE in some other test. Multi-condition coverage is a stronger requirement than branch coverage.
- Relational coverage: This checks for tests that probe common mistakes regarding relational operators. A likely mistake could be using “>” when “>=” is intended.

III. DESCRIPTION OF THE EXPERIMENT.

Structure testing is performed in a simulated environment at module level.

Research Theme: To calculate the fault detection rate of structure testing depicting the effectiveness of structure testing.

Research Method: Simulation

Phase 1

- First of all we create an example unit.
- Then all test cases are generated.
- After that test cases are executed. With the help of test case result generator we will test which test cases are valid and which are invalid. In this way erroneous test cases get detected.

Phase 2

In this phase defect detection rate is calculated as follows:

$$\text{Fault Detection Rate} = \text{TestRate} \times \text{TestMethodDD}$$

Where,

$$\text{TestMethodDD} = \text{number of defects/KLOC}$$

$$\text{TestRate} = \text{TestMethodTR} \times \text{TesterResource}$$

Where,

$$\text{TestMethodTR} = \text{KLOC/day}$$

$$\text{TesterResource} = \text{number of testers}$$

Table 1

Unit Under Test	Parameters	Counter
Module A	Lines Of Code	337
	Faults	2
Module B	Lines Of Code	324
	Faults	2
Module C	Lines Of Code	325
	Faults	10

- The first test for module A will test the functionality of the built in product block. Product block requires two inputs and delivers one output. Inputs to the block are random vectors. The input is a cell containing the separate inputs to the test case. Now we make the first test case: input, output and comparator. In this case we want the product to be exactly accurate.
- The second test is basically identical to the first except that we test the sum block.
- In the third test a general adder block is designed which can take arbitrary number of inputs and add them together. In this case we have used 20 inputs. We also used the “approximately equal” comparator since the block can make small rounding errors.

The calculations we have performed are as follows:

$$\text{TestMethodTR for Module A} = 0.337/1 = 0.337$$

$$\text{TestMethodTR for Module B} = 0.324$$

$$\text{TestMethodTR for Module C} = 0.325$$

$$\text{TestRate for Module A} = 0.337 \times 1 = 0.337$$

$$\text{TestRate for Module B} = 0.324$$

$$\text{TestRate for Module C} = 0.325$$

$$\text{TestMethodDD for Module A} = 2/0.337 = 5.93$$

$$\text{TestMethodDD for Module B} = 2/0.324 = 6.17$$

$$\text{TestMethodDD for Module C} = 10/0.325 = 30.76$$

$$\text{Fault Detection Rate} = \text{TestRate} \times \text{TestMethodDD}$$

$$\text{Fault Detection Rate for Module A} = 0.337 \times 5.93 = 1.998$$

$$\text{Fault Detection Rate for Module B} = 0.324 \times 6.17 = 1.999$$

$$\text{Fault Detection Rate for Module C} = 0.325 \times 30.76 = 9.997$$

IV. ANALYSIS OF THE RESULT.

In this simulated experiment we have done the analysis of fault detection rate of structure testing. Above experiment is performed at unit level in which we have taken three modules.

As the number of defects in three modules vary, the fault detection rates also differ.

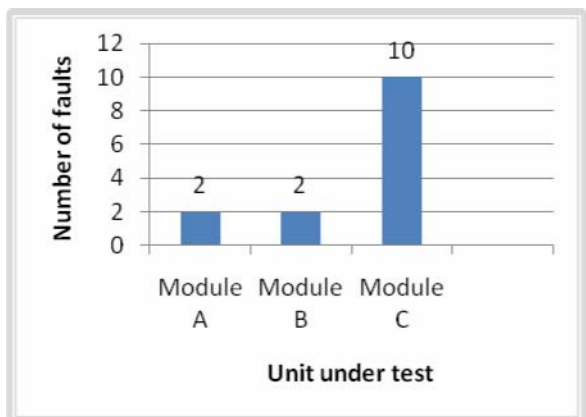


Fig 2: Faults detected in unit under test

Table 2 present the summary descriptive statistics of fault detection effectiveness for structure testing.

Testing Technique	Unit Under Test	TestRate	TestMethodDD	Defect Detection Rate
Structure Testing	Module A	0.337	5.93	1.998
	Module B	0.324	6.17	1.999
	Module C	0.325	30.76	9.997

CONCLUSION

In this paper we have shown the results of simulated experiment which help us to analyse fault detection rate of structure testing. We consider this experiment as a starting point and would like to extend this to include type of faults detected metric with the combination of more than two testing techniques.

REFERENCES

[1] Antonia Bertolino. “Software Testing Research: Achievements, Challenges, Dreams.” Future of Software Engineering(FOSE’07), 2007.

[2] Proceedings of the twenty-second IEEE/ACM international conference on automated software engineering. Atlanta, Georgia, USA, SESSION: Testing Pages: 343-352, 2007.

[3] Oliver Laitenberger, “Studying the Effects of Code Inspection and Structural Testing on Software Quality” Fraunhofer Institute for Experimental Software Engineering.

[4] Boris Beizer., “Software Testing Techniques”. International Thomson Publishing Inc., 2nd edition, 1990.

[5] “A survey of coverage based testing tools” ACM, AST’06, May 23, 2006, Shanghai, China, 2006.

[6] Per Runeson, Anneliese Andrews, “Detection or Isolation of Defects? An Experimental Comparison of Unit Testing and Code Inspection” in IEEE Proceedings of the 14th International Symposium on Software Reliability Engineering (ISSRE’03).

[7] Erik Kamsties and Christopher M. Lott, “An Empirical Evaluation of Three Defect-Detection Techniques” in the Proceedings of the Fifth European Software Engineering Conference, 26–28 September 1995.

[8] Institute of Electrical and Electronics Engineers. Standard Glossary of Software Engineering Terminology, 1983.

[9] Sommerville, I., Software Engineering (7th ed.), Addison-Wesley, 2004.

[10] Ng, S. P., Murnane, T., Reed, K., Grant, D. and Chen, T. Y., “A Preliminary Survey on Software Testing Practices in Australia”, Proceedings of 2004 Australian Software Engineering Conference, pp. 116-125, 2004.