International Conference on "Latest Innovations in Science, Engineering and Management"
Indian Federation of United Nations Associations, New Delhi, India ICLISEM - 16
1st October 2016, www.conferenceworld.in ISBN : 978-93-86171-08-5

MUTATION TESTING: AN ERROR SEEDING SOFTWARE TESTING TECHNIQUE

Deeksha Gupta

Department of Computer Science and Applications, MCM DAV College for Women, Chandigarh

ABSTRACT

Software testing is a key component of software development process. And due to importance of testing in a software life span it has got most attention from research point of view. Mutation testing is one of the white box testing which is very fascinating to researcher due to its approach to improve quality of software. This paper explains the basic concept behind mutation testing, process of mutation testing, its various types and advantages and disadvantages of mutation testing.

Key words: mutation testing, fault based testing technique, strong mutation testing, Selective mutation testing, mutant

I. INTRODUCTION

Presence of bugs in software can be very dangerous. It was the existence of bugs in software that had led to China Airlines Airbus crashing [1], Canada's Therac-25 radiation therapy machine malfunction [2] and failure of a $1.2 billion military satellite launch [3] So It is very important to detect and remove the bugs and any anomaly in the software. The process of evaluating software and its components to find errors is called software testing. Although testing increases the effort and cost of building software but still it is integral part of every phase of software development because testing ensures the correctness**,** completeness, reliability and quality of software. Software testing is done to detect various errors for different aspects of the software like functional, user interface, usability, performance, security etc. Software Testing ensures whether the software satisfies its specified functional and nonfunctional requirements or not. Software testing techniques are broadly divided into two categories white box testing and black box testing. One of the eminent white box testing in mutation testing [4].This paper covers the need of mutation testing, basic concept of mutation testing, its types and pros and cons of mutation testing.

II. NEED OF MUTATION TESTING

Mutation testing is fault based testing normally used for unit testing. In this testing technique the software is tested to check the completeness of test suite which in turns ensures the quality of software [5]. In Mutation testing simple bugs are introduced in the program to check the adequacy of the test suite. If test suite fails to identify the seeded faults then effective test cases are added to it to make it sufficiently strong.

In conventional testing techniques the software is tested against structural test suite. So the verification of correctness of a program depends upon the efficiency of test cases. Test suite should be rich enough to include test cases for detecting all possible defects in program. The objective of mutation testing is to find the flaws of test suite and then modifying suite to ensure its reliability in finding errors. It helps to create correct and efficient test cases. Mutation testing makes test suit robust so that all possible defects can be identified by it.

Mutation test is an effective technique to ensure the adequacy of test suite. The basis of mutation testing technique is two main assumptions: First competent programmer hypothesis, Second the coupling effect. Mutation testing is based on two assumptions: the competent programmer hypothesis and the coupling effect. The competent programmer hypothesis supposes that although program is written by skilled programmer but it may not be error free. It may contain very small error that may deviate program output of program from the intended one. The coupling effect is based on fact that detection of small errors may cause the identification of big faults. That is simple errors in a program may be associated with complex error.

III. MUTANTS

Mutants are the key components of the mutation testing. A mutant is version of original program under test in which simple bug is added intuitively. Each mutant contains one simple error. The success of mutation testing depends on the number of mutants generated. On the basis of concept of mutant generation, Mutants can be of different types:

Syntactical mutants: Mutants that are generated by making change in syntax of the program. There mutants can be detected by the compiler. For example: x=zy++.

Minor mutants: Mutants that can be detected by any test case of the test suit.

Equivalent mutant: These mutants are not detected by any test case because, these are not actually errors. These mutants produce the same output as that of original program.

Value mutant: these mutants are generated by changing the value of constants and variable across the boundary values that is by replacing values to either too large or too small numbers.[6]

Condition Mutants: These mutants are generated to check the efficiency of test cases related to decision control statements accuracy. This can be done by replacing arithmetic, relational or logical operators in conditions. [6]

Statement mutants: In these mutants the statements are removed, replaced or duplicated to check the efficiency of test cases.

From the point of testing the value mutants, condition mutants and statement mutants are more useful as they help to find the inefficiency of test cases [7].

IV. PROCESS OF MUTATION TESTING

Mutation testing involves generation of mutants, testing and analyzing the outcomes. The whole process can be implemented by following the given steps:

Step 1: Generation of mutants: For the program under test various mutants are generated. These mutants may be generated by introducing errors by replacing any operator, operand or statement of the program.

Step 2: Testing: In second step the original program as well as the generated mutant is tested against all test cases of test suite

Step 3: Comparison of test outcome: Now the outcome of mutant program is tested with that of original program. If outcome is different, then mutant is killed that is it is not further tested with rest of the cases of test suite. It interprets that test suite is robust enough to handle the particular fault added in the killed mutant [6].

Step 4: Updation of test suite: In the previous step if the outcomes of mutant and original programs is same for all the test cases of test suite it may further have two interpretations. One the mutant is equivalent mutant of the original program. An equivalent mutant is a version of original program that has different syntax but same semantic as that of original program.[8] Two, the test suite is not adequate to handle that particular fault so more effective test case is added to test suite so that one particular fault can be identified by testing.

These steps are repeated for all mutants and for each mutant for all test cases in test suite. But this testing should be stopped if specified reliability of test suit has been achieved or if pursuing further in testing is resulting in much testing cost as compared to benefit from it.

Fig 4.1 represents the process of traditional mutation testing:

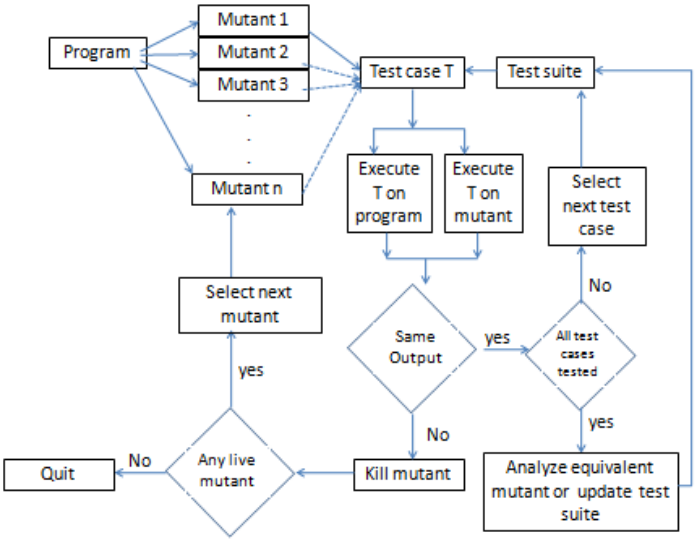

Fig. 4.1 Mutation Testing Process

V. VARIANTS OF MUTATION TESTING

On the basis of the time when the results of tests on mutated and original module are compared, mutation testing is divided into three categories:

- **Strong mutation testing:**

It is traditional mutation testing. In strong mutation testing a mutant is killed if following specified conditions are satisfied:

1. Test case should execute the mutated statements of mutated version.
2. Then outcome of entire module of original and mutated versions are compared.

So in Strong mutation testing the outcomes of test case are compared after the execution of whole module. So strong mutation testing covers code, path and control testing. It is more powerful as it finds those missing test cases that cannot be revealed by weak mutation testing. This testing is very time consuming and expensive in terms of computational effort [4].

- **Weak mutation testing:**

In weak mutation testing the mutant is killed after following the given conditions:

1. One test case should be executed for the replaced statement in mutant.
2. The outcome of internal state of mutated program is different from output of internal state of original program.

If both of the above mentioned conditions satisfy and then mutant is killed then the testing is called weak mutation testing. That is in weak mutation testing the outcomes of tests are compared just after testing each component of the mutated module.

Weak mutation testing follows the code coverage method. Control coverage is not covered in it. In this type of testing it is not required to execute the entire mutated program, just run it to the mutated statement and then compare the outcome by comparing the internal states of the mutated and original program. It takes less computing time, less computation and also provides weal reliability of test suite [6].

- **Firm mutation testing:**

The basic concept of firm mutation testing is to conquer the limitations of weal mutation testing and string mutation testing. In this technique the time for comparison of outcome of mutated program and original program is in between that of weak and strong mutation testing. This technique is evolved to give the best of both weak and strong testing. This takes less time and computational effort as compared to strong mutation testing and the test suite is sufficiently updated to give a better performance as compared to weak mutation testing [9].

- **Selective Mutation testing**:

In mutation testing the complexity increases as the number of mutants increases. Mutants are generated with small change in the syntax of program. These small changes in syntax are specified by the mutation operator. The basic concept of selective testing is to reduce the set of these mutation operators to reduce the number of generated mutants. In this testing first a subset of mutant operators is identified that generate a subset of all possible mutants. Only few of these operators are used to generate a relatively smaller subset of mutants. As the number of mutants reduces, the time cost and effort cost of mutation testing also decreases [10].

- **Mutant Clustering**:

This testing is based upon the idea of clustering of mutants on the basis of test cases. In this technique first all the possible mutants are generated after that clustering is done. If different mutants can be detected by same test cases then all those different mutants are grouped together in a cluster. This reduced the testing effort as only

selected mutants from a cluster are used for practical implementation of mutation testing. So it reduces the complexity of mutation testing [11].

- **Higher Order Mutation**:

First order mutant are generated by using mutant operators only once in a mutant. Higher Order Mutants are created by using mutant operators more than one time. A higher order mutant replaces no. of first order mutants. So a smaller set of higher order mutants give better result in order to test the adequacy of test suite. These higher order mutants are difficult to kill, so this technique is efficient in terms of computation effort and time taken to complete the test suite [12].

VI. ADVANTAGES OF MUTATION TESTING

- **Full code coverage**: this testing is implemented on all the statements, paths and conditions of source code. So the entire source code is covered by this technique.
- **Unambiguous source code**: testing finds and removes all the ambiguities present in the source code.
- **Improved quality test suite**: Mutation testing provides a robust test suite which is mutant adequate with 100% mutant score [13].
- **Improved quality software**: The main focus of mutation testing is to improve quality of test cases which in terns improves the quality of source code.
- **Effective test case identification**: This method helps to create effective and correct test cases to find the defects in a program.
- **Easy debugging**: Debugging of source code is easy as compared to debugging in conventional testing.
- **Availability of auto mated tools**: Various powerful automated tools are available to perform mutation testing in different environments. These automated tools are available off the shelf. Use of these automated tools for testing fastens up the process and provided component based development [13][14].

VII. DISADVANTAGES OF MUTATION TESTING

- **Difficult implementation**: It is difficult to implement mutation testing manually. Use of any automated tool makes it easier to implement [15].
- **Expensive and time-consuming**: Generations of too many mutants and their testing makes this testing time consuming and expensive in terms of effort and computation required [14].
- **Needs skilled tester**: This technique requires skilled tester with sound knowledge of testing as well as programming.
- **Limited applicability**: This method cannot be applied for black box testing because it requires changing the source code [8].
- **Large population of mutants:** Even for a small program, a large number of mutants can be generated. Generation of all possible mutants and their testing makes this technique very complex [9].

- **Unfirmation of equivalent mutants:** The Checking for the equivalence mutants is quite difficult and complicated [15].

VIII. CONCLUSIONS

Testing is an important part of software development. It is not about just debugging. In mutation technique, the mutants are generated by seeding the errors in program under test. Test cases are selected and generated mutants are tested against selected test cases. It consumes time, computational effort and money. So various alternative approaches are developed for traditional mutation testing. Software testing is an art. Mutation testing makes test cases so strong so that only a reliable program can pass the test suite.

IX. FUTURE ASPECT

The time and cost are the biggest limitation of mutation testing. Although improved techniques of mutation testing like selective, mutant clustering mutation testing can be used to overcome these limitations. But in the dynamic environment of computer science mutation testing can be improvised so as to support systems with different software and hardware support. A lot of work can be done in this field to make new and powerful automated tools for mutation testing.

REFERENCES

[1] http://www.sozogaku.com/fkd/en/cfen/CA1000621.html

[2] Israelski, Edmond W., and William H. Muto. "Human factors risk management as a way to improve medical device safety: a case study of the therac 25 radiation therapy system." *The Joint Commission Journal on Quality and Patient Safety* 30.12 (2004): 689-695.

[3] http://www.guru99.com/software-testing-introduction-importance.html

[4] Singh, P.K, Sangwan O.P. and Sharma A., "A Systematic Review on Fault Based Mutation Testing Techniques and Tools for Aspect-J Program", proceedings of *3rd IEEE International Advance Computing Conference, IACC-2013*, India, 22-23 Feb. 2013.

[5] Fraser G, Zeller A. ,"Mutation-driven generation of unit tests and oracles", Proceedings of the *19th International Symposium on Software Testing and Analysis (ISSTA'10*), ACM: Trento, Italy, 2010; 147–158.

[6] Anbalagan P. and Tao X., "Automated Generation of Pointcut Mutants for Testing Pointcuts in AspectJ Programs", Proceedings of the *19th International Symposium on Software Reliability Engineering (ISSRE'08)*, IEEE Computer Society, pp. 239–248, 11-14 November 2008

[7] B. K. Aichernig and C. C. Delgado, "From Faults Via Test Purposes to Test Cases: On the Fault-Based Testing of Concurrent Systems," Proceedings of the *9th International Conference on Fundamental Approaches to Software Engineering (FASE'06)*, ser. LNCS, vol. 3922. Vienna, Austria: Springer, 27-28 March 2006, pp. 324–338.

[8] Grun B.J., Schuler D., Zeller A., "The Impact of Equivalent Mutants", Proc. Fourth International Workshop Mutation Analysis, pp. 192-199, Apr. 2009.

[9] A. J. Offutt, J. Pan, K. Tewary, and T. Zhang. "An Experimental Evaluation of Data Flow and Mutation Testing". *Software-Practice and Experience*, vol. 26(2): 165-176, February 1996.

[10] Duncan, I. M. (1993). "Strong mutation testing strategies". Ph. D. thesis, Department of Computer Science, University of Durham.

[11] Bluemke, Ilona, and Karol Kulesza. "Reductions of operators in Java mutation testing." *Proceedings of the Ninth International Conference on Dependability and Complex Systems DepCoS-RELCOMEX. June 30–July 4, 2014, Brunów, Poland*. Springer International Publishing, 2014.

[12] Byoungju, Choi, and Aditya P. Mathur. "High-performance mutation testing".*Journal of Systems and Software* 20.2 (1993): 135-152.

[13] https://www.tutorialspoint.com/software_testing_dictionary/mutation_testing.htm

[14] Gupta, D., "Component Based Software Engineering: A Customized Approach for System development", *International Journal of Innovative Research in Science and Engineering*, Vol. 2, issue 9, 2016,64-72

[15] Andrews JH, Briand LC, Labiche Y. "Is mutation an appropriate tool for testing experiments?" Proceedings of the *27th International Conference on Software Engineering (ICSE'05)*, St Louis, Missouri, 2005; 402–411