ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Issue 8, August 2017

A Deep Survey on Mutational Testing and Test Adequacy Check

J Santhosh[1], Mini N[2]

Assistant Professor, Department of Computer Science, Sree Narayana Guru College, K G Chavadi, Coimbatore, Tamil Nadu, India [1]

M.Phil Research Scholar, Sree Narayana Guru College, K G Chavadi, Coimbatore, Tamil Nadu, India [2]

ABSTRACT: The Mutation Testing is a fault based software testing technique that has been widely studied so far. This helps to generate the effective test cases and it provides a testing criterion. The mutation adequacy testing and score can be used to test the effectiveness of the test cases. It is capable to detect the faults and mistakes in the program as well. For successive future direction, this is necessary to analyze and compare the earlier works on the mutation testing and test adequacy criterion check. This paper provides the comprehensive analysis of mutation testing approaches, tools, techniques and its results. These analyses give evidence that Mutation Testing techniques and tools are reaching a state of maturity and applicability, while the topic of Mutation Testing itself is the subject of increasing interest.

KEYWORDS: Software Testing, Mutation Testing, Test adequacy criterion

I. INTRODUCTION

Software testing is the process of seeking the software defects before the launch in the market. This is the most important process in software development life cycle. In the software testing domain, mutation testing is the common and popular testing method, which can be used to evaluate the earlier test cases. This type of testing always helps to improve the test case generation process [1]. A mutant is generated by making changes in the original program. These syntactic changes are performed by certain rule, which is called as mutation operator. If a test case is able to differentiate the outputs between a mutant and its original program, the mutant is said to be *killed.* A mutant is equivalent, if it cannot be killed by any test case. Generally, the adequacy of mutation testing named *mutation score* is defined as the ratio of the number of killed mutants to the total number of non-equivalent mutants. This paper provides an overview of mutation testing and the recent researches made on mutation testing and its test adequacy criterion verification. This survey helps to find the problems in the recent research and gives an idea for the future development.

A. Mutation Testing

Software testing involves two types of testing; black box and white box testing. Black box testing is concerned with input-output behavior or functionality of the component, whereas white box testing deals with the internal program structure. For black-box testing, the specification of the component has to be systematically manipulated to generate test cases. The work is focused on specification-oriented mutant generation to validate the behavioral specification of the component. In case of white-box testing, this can be done by making systematic changes in the source code of the component. In both the cases testing shows that a program satisfies its test data but cannot assure the quality of test data. Mutation testing is an accepted technique for improving the quality of developed software. It is a white box testing technique that is applicable during unit testing of program and involves creation and execution of different versions known as mutants of a program. Mutation testing attempts to measure and improve the quality of a test suite. It is a fault based testing method to increase the testability of the component and can be applied to various objects, specifications, classes, contracts and interfaces. Mutation testing, which is introduced in [2], is a widely accepted fault-based testing technique that determines test adequacy by measuring the ability of a test suite to

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Issue 8, August 2017

discriminate the program from some alternative program (mutants). However, the huge computational cost caused by executing thousands of mutants and detecting plenty of equivalent mutants manually is an obstacle of applying mutation testing in practice. Mutants are created by introducing a single, small, legal syntactic change in the software. Given a program S and corresponding test suite T, the first step of the mutation testing is to construct the alternative programs that is, mutants, (S). Then each mutant and the original program S have to be executed against each test case. If the output of mutant M remains the same as the output of the original program S, consider M as alive; otherwise, M is considered to be dead or distinguished. M remaining alive can be caused from the two factors such as inadequate test data and the mutant is equivalent to the original program. If the mutant is equivalent to the original program, then it will not contribute to the adequacy of T and should be eliminated. After eliminating the equivalent mutants, the test adequacy check is performed. The number of the remaining mutants that are detected by the test suite can be used to evaluate the corresponding test adequacy. This is formally defined as mutation score:

$$MutationScore = DeadMutants / (TotalMutants - EquivalentMutants)$$

With the concept of subsuming the single faults injected in source codes to construct higher order faults in the program, mutants are classified into two types: First Order Mutants (FOMs) and Higher Order Mutants (HOMs) (Table 1.0).

Table 1.0: First Order and Higher Order Mutants

Original Program S	**First Order Mutant (FOMs)**	**Higher Order Mutant (HOMs)**
if(a>0 && b>0) return 1;	if(a>0 || b>0) return 1;	if(a<=0 && b<=0) return 1;

FOMs are generated by applying single order mutation operators. HOMs is generated by applying more than one mutation operators. A combined HOM is harder to kill than the FOMs from which it is constructed. Therefore, it may be preferable to replace the FOMs with the single HOM. Consider a strongly subsuming higher order mutant, H, constructed from FOMs T1… Tn. The set of test cases that kill H also kill each and every first order mutant T1…Tn. As a result, H can replace all of the mutants T1…Tn without loss of test effectiveness. The converse does not hold that is, there exist test sets that destroy all FOMs T1…Tn but which fail to kill H. Higher order mutants reduce the number of the mutants generated as well as the number of test cases to kill these higher order faults. This way they improve the efficiency of the mutation testing as well make the mutation testing practically feasible.

B. Association between Mutation testing and Test suite

When a mutant program is executed against the test suite, there could be three different possibilities, depending upon which one can determine the effectiveness of test suite and the efficiency of mutation operators. Moreover for the test case to kill the mutant, the following two conditions should be met:

I. Test input data should cause different program states for the mutant and the original program.

II.. The variable that is affected by change should be propagated as output and be checked.

If the test suite fails that is, it is able to detect the change between the program and its mutant; it shows that the test suite is sufficient. If all the test cases pass, it indicates weakness in the test suite which needs to be repaired by adding more test cases.

Weak Mutation Testing requires that only the first condition is satisfied. It is closely related to code coverage methods and requires much less computing power.

Strong mutation testing requires that both conditions be satisfied. Strong mutation is more powerful, since it ensures that the test suite can really catch the problems.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Issue 8, August 2017

Mutation Operator	Description
AOR (+, -, *, /)	Arithmetic Operator Replacement e.g. + can be changed with -
ROR (<, >, <=, >=, ==,!=)	Relational Operator Replacement e.g. greater than is replaced with less than

Table 2.0: Sample Mutation Operators

Mutants can be placed anywhere in the source code, mutation coverage is equivalent to structural coverage. A higher order mutant is a program that can be obtained by applying several operations from a set of first order mutant operations [3]. The relationship between simple faults and complex faults can be investigated. Simple faults are the first order mutants while complex faults are higher order mutants. Higher Order Mutation Testing involves more than one mutation position within a component or program. An application program has been designed to automate the creating of mutants of single order as well as higher order using arithmetic and relational mutant operators. In the initial step of mutant generation, first order and higher order mutants are created. Though number of mutation operators is available some sample mutation operators are shown in Table 2.0. After the mutant generation, the system will be reevaluated.

II. LITERATURE REVIEW

This section gives the study of the recent researches on the mutation testing. A weak mutant testing method [4] was proposed in the earlier research. This checks the whole program for mutant by checking the mutated statement immediately after the execution. In strong mutation testing, a mutant is killed when its output differs from that of the original program. But in weak mutant testing, this scenario is different.

In paper [5] authors provided an empirical evaluation of the mutation testing techniques and its applications. The certification attributes should be satisfied in the part of testing, so authors applied mutation testing using the high integrity subsets of C. the authors also analyzed the root causes of the failures in the test cases. They also found the effective mutant type with effective coding suggestions. The authors examined the relationships between the program features with the mutation survival. This considers the typical verification life cycle with the real time software's. The highlighted features of this paper are, it supports two types of languages such as C and Ada. Initially the source code is analyzed with LOC (lines of Code) and CC (Cyclomatic Compelxity) Then generation phase detects the syntactically incorrect mutants from the total mutants, the retested and equivalent mutants are gathered to find the final score.

In paper [6], authors specified the impact of inadequate test cases and ineffective defect detection. The authors surveyed the mutation testing with cost reduction techniques, which is an important to determine the tool suitability. From this paper, the important features for the mutation testing tool are identified.

In paper [7], authors developed an algorithm to define the variables and branches for mutation testing. This involved the variables and branches as a metric, and this is named as FunctionRank, this finds the relative importance from the application behavior and ranks the functions according to that. The authors guide the mutation generation process by leveraging the static and dynamic analysis. This is implemented in the JavaScript language. Authors implemented the approach in MUTANDIS tool, which provides 93% of the non equivalent mutants and 75% of results are in the top ranked functions. However, since the authors considered only 100% adequate test suites for the mutation adequacy criteria, which is not reliable for all applications. This considers the 100% relationship between the percentage of the mutation adequacy score and the fault detection effectiveness. These two factors are not fully investigated in this paper.

In paper [8], a new technique is proposed to reduce the test case execution cost. In configuration aware structural testing method, the number of test cases is usually high. To minimize this, authors proposed a combinatorial optimization technique and the optimization technique. Initially the combinatorial optimization is used. This generates

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Issue 8, August 2017

an optimized test suite by the sample input configurations. For further optimization, the adaptive mechanism by using the mutation testing is used. This utilized the cuckoo search with the combinatorial approach to filter the test cases. However, the system performance is not adequate and not suitable for all platforms.

In paper [9], authors aimed to decrease the mutants by deploying a novel method of reducing mutants based on dominance relation. In the deployed novel method, another program is shaped by embedding mutant branches into the first program. The proposed strategy in the paper is connected to test ten benchmark projects and six classes from open-source ventures. The demonstrations of the technique achieved more than 80% mutants are effective. The authors concluded that the efficiency of mutation testing is greatly improved by the method.

In paper [10], authors proposed a new evolutionary mutation test, which is based on the genetic algorithm. This performs the automated test case generation process to reduce the mutation testing computational overhead. The authors have developed a fitness function to evaluate the test case fitness by modifying the object state in every iteration. It is more capable to improve the test case. The author used eMuJava tool for experiment. The experiments of this concept are performed eMuJava v2.0. The genetic algorithm has the random population selection method, which incurs high number of iterations.

In paper [11], authors presented two novel approaches for the automated testing, which were written in Alloy language. This introduces the automated test generation that creates the test cases for mutation testing. This also identifies the equivalent mutants using SAT.

In paper [12], authors performed the theoretical and empirical study on diversity-aware mutation adequacy criterion. This is defined as distinguishing mutation adequacy criterion. This approach is fully satisfied when each of the considered mutants can be identified by the set of tests that kill it, thereby encouraging inclusion of more diverse range of tests. The author evaluated the test cases and its fault detection capabilities.

Table 3.0 Mutation testing related researches comparison table

Paper id	Title	Authors	Description	Merits	Demerits
5	An empirical evaluation of mutation testing for improving the test quality of safety-critical software.	Baker, Richard, and Ibrahim Habli.	Evaluate mutation testing	Authors found root cause for the test case failure. Found most effective mutant type.	Safety-critical software's are not used for the evaluation. determining equivalent mutant behavior is still a manual overhead
6	Mutation Testing	Reales, Pedro, Macario Polo, Jose Luis Fernandez-Aleman, Ambrosio Toval, and Mario Piattini.	Analyzed the features of effective mutation testing with the cost effective techniques	Authors gives the survey related to the cost effective features	The mutation testing techniques are reviewed rather than any contribution. The proof for the analysis is not adequate.
7	Guided mutation testing for javascript web applications	Mirshokraie, Shabnam, Ali Mesbah, and Karthik Pattabiraman.	Developed Guided Mutation Algorithm to define the variables and branches for mutation testing	Exhibits high structural complexity. minimize the number of generated mutants	Considered only 100% adequate test suites. Not reliable for all the applications

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Issue 8, August 2017

8	Test case minimization approach using fault detection and combinatorial optimization techniques for configuration-aware structural testing	Ahmed, Bestoun S	Combinatorial optimization and sampling technique proposed.	Search for an optimal solution - useful for different testing techniques	Performance is not adequate
9	Mutant reduction based on dominance relation for weak mutation testing	Gong, Dunwei, Gongjie Zhang, Xiangjuan Yao, and Fanlin Meng.	A novel approach is proposed to obtain the non-dominated mutant branches which correspond to the mutants after reduction.	Time spent in generating test data is shortened.	Only suitable for reducing mutants generated by traditional (method- level) mutation operators and not suitable for class-level operators. Scalability issues.
10	Improved Genetic Algorithm to Reduce Mutation Testing Cost	Bashir, Muhammad Bilal, and Aamer Nadeem.	Genetic algorithm with effective fitness function for mutation testing is proposed.	Evaluates the test case fitness	Huge in size of iterations.
11	Automated Test Generation and Mutation Testing for Alloy	Sullivan, Allison, Kaiyuan Wang, Razieh Nokhbeh Zaeem, and Sarfraz Khurshid.	Presented two novel approaches for automated testing of models written in Alloy	Effective way to validate the quality of Alloy models. Automatic test generation.	Not suitable for all languages.
12	A Theoretical and Empirical Study of Diversity-aware Mutation Adequacy Criterion	Shin, Donghwan, Shin Yoo, and Doo-Hwan Bae.	Calculates the mutation adequacy score and evaluates the test cases.	Stronger mutation adequacy criterion is promising	Need extra study on cost-benefit analysis. test data generation is not performed in this paper.

The table 3.0 shows the overall descriptions of the recent studies on mutation testing along with the advantages and disadvantages of each approach. This survey considers only the recent papers on the mutation testing and its branches. This includes mutation test cost effective techniques, adequacy criterion check, automated test case generation in mutation testing etc., there are numerous technique and approaches used in the literature, however the most of the techniques used to evaluate the cost reduction techniques on a specific language. And few approaches test the test cases

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Issue 8, August 2017

for its adequacy criterion. The test case data generation with additional features can improve the approaches. From the analysis, the mutation program from genetic approach yielded better result. But there are certain limitations were found in the base paper [12].

III. CONCLUSION

This paper gives a detailed survey and analysis of Mutation Testing and its adequacy criterion. The paper covers the basic information about the mutation testing and surveys various approaches related to that such as, optimization techniques, equivalent mutant detection, applications etc. There has been much enhancement to decrease the cost of the Change Testing process. The paper additionally discovers confirmation that there are an expanding number of new applications. There are increasingly, bigger and more sensible projects that can be taken care of by Mutation Testing. Late patterns additionally incorporate the arrangement of new open source and mechanical instruments. These discoveries give proof to help the claim that the field of Mutation Testing is undergoing vast changes. The study highlighted some shortcomings of the recent works and this can be enhanced in the future work.

REFERENCES

[1]. G. Fraser , A. Zeller ,Mutation-driven generation of unit tests and oracles, IEEE Trans. Softw. Eng. 38 (2) (2012) 278–292.
[2]. DeMillo, Richard A., Richard J. Lipton, and Frederick G. Sayward. "Hints on test data selection: Help for the practicing programmer." *Computer* 11.4 (1978): 34-41.
[3]. Nguyen, Quang Vu, and Lech Madeyski. "Problems of Mutation Testing and Higher Order Mutation Testing." In *ICCSAMA*, pp. 157-172. 2014.
[4]. Bhatia, Vasundhara, and Abhishek Singhal. "Design of a Fuzzy model to detect equivalent mutants for weak and strong mutation testing." *Information Technology (InCITe)-The Next Generation IT Summit on the Theme-Internet of Things: Connect your Worlds, International Conference on*. IEEE, 2016.
[5]. Baker, Richard, and Ibrahim Habli. "An empirical evaluation of mutation testing for improving the test quality of safety-critical software." *IEEE Transactions on Software Engineering* 39, no. 6 (2013): 787-805.
[6]. Reales, Pedro, Macario Polo, Jose Luis Fernandez-Aleman, Ambrosio Toval, and Mario Piattini. "Mutation Testing." *IEEE software* 31, no. 3 (2014): 30-35.
[7]. Mirshokraie, Shabnam, Ali Mesbah, and Karthik Pattabiraman. "Guided mutation testing for javascript web applications." *IEEE Transactions on Software Engineering* 41, no. 5 (2015): 429-444.
[8]. Ahmed, Bestoun S. "Test case minimization approach using fault detection and combinatorial optimization techniques for configuration-aware structural testing." *Engineering Science and Technology, an International Journal* 19.2 (2016): 737-753.
[9]. Gong, Dunwei, Gongjie Zhang, Xiangjuan Yao, and Fanlin Meng. "Mutant reduction based on dominance relation for weak mutation testing." *Information and Software Technology* 81 (2017): 82-96.
[10]. Bashir, Muhammad Bilal, and Aamer Nadeem. "Improved Genetic Algorithm to Reduce Mutation Testing Cost." *IEEE Access* 5 (2017): 3657-3674.
[11]. Sullivan, Allison, Kaiyuan Wang, Razieh Nokhbeh Zaeem, and Sarfraz Khurshid. "Automated Test Generation and Mutation Testing for Alloy." In *Software Testing, Verification and Validation (ICST), 2017 IEEE International Conference on*, pp. 264-275. IEEE, 2017.
[12]. Shin, Donghwan, Shin Yoo, and Doo-Hwan Bae. "A Theoretical and Empirical Study of Diversity-aware Mutation Adequacy Criterion." *IEEE Transactions on Software Engineering* (2017).