An Oversight on Mutation Testing

Deepika Sharma
Research Scholar, Department of Computer Science and Applications, Kurukshetra University, Kurukshetra, Haryana
Email: lakhanpal_deepika@yahoo.in
Sanjay Tyagi
Assistant Professor, Department of Computer Science and Applications, Kurukshetra University, Kurukshetra, Haryana
Email: tyagikuk@gamil.com

Abstract
Traditional testing methods only measure the execution of code. They do not take into account the actual detection of faults in the executed code. It is therefore only able to test the execution of code and not the faults. Mutation testing is an important method of fault revealing. The main aim of mutation testing is to test the quality of test cases in such a way that it should be able to fail the mutant code. Mutation testing also called as fault revealing strategy because here faults are introduced in the program and then different test cases are applied on the mutant to find bugs in the mutated program. It is the process of rewriting the source code by introducing changes in small ways to remove the redundancies in the source code. This paper reviews the mutation testing concept, effectiveness of test cases, categorizes the mutant and focus on the techniques to reduce mutant.

Keywords: Fault revealing, Mutant, Mutation Testing, Test case.

1. Introduction

Software testing is very important phase among all phases such as requirement analysis, design, and implementation etc in software development life cycle. Out of this, software testing plays a very crucial role in software development. If the testing of software is not appropriate, then there is a tradeoff in the quality of the software product. There are generally three main type of testing such as unit testing(in which each component of the product will be tested), integration testing(in which some of components are integrated and then testing will be performed) and system testing(in which whole system is tested before deliver it to the customer)[1]. Here our focus is on unit testing. If unit testing doesn't find any bug or error, it does not mean that there aren't any bugs in the program. For this, mutation testing is used to test your test cases.
Mutation testing evaluates the quality of *existing* software tests. The idea is to modify (mutate) code in a small way and check whether the existing test set will detect and reject the change. If it doesn't, it means that the tests do not match the code's complexity and leave one or more of its aspects untested. Detecting and rejecting such a modification by the existing tests is known as *killing* a mutant. It is a structural or fault based testing, which uses the structure of the code by introducing some faults in the code to guide the testing program. Mutation was originally proposed in 1971 and the research on mutation testing was implemented by Timothy Budd in 1980 [2], but it will not become popular at that time due to high cost involved. But now again it has been opted for testing the quality of test cases, and widely used for various languages such as java, xml. According to Budd "Mutation testing is a fault based testing technique in which we seed the errors in the program and find the errors" [2].
This paper is organized as follows: Section II introduces the mutation testing, describes how to calculate the mutation score after killing the mutant, effectiveness of test cases and categorizes the mutants. Section III discusses about the related work in this field. Section IV is about equivalent mutant which is a halting problem. Section V discusses about the

techniques to reduce mutants. Section VI brings the conclusion and future scope of the paper.

2.Mutation Testing

Mutation testing is a method of introducing errors in the program for accessing the quality of test cases was proposed by the Hamlet [3]. In mutation testing, some changes are introduced in the program called the mutated program and then the test cases are applied on the mutated program to analyze the output as whether it is able to find error or not. According to competent programmer hypothesis the programmer always commit small mistakes during coding. That's why only simple errors will be introduced in the program and if test cases are able to find simple error then it can also able to find much more complex error. Mutants are generated by using different operators in the program known as mutant operators [4] or by changing any statement of the program. As in fig 1, Mutants are generated by seeding some faults in the program and then test cases will be executed on the mutant program. If the test is able to find errors in the mutated program or produce different output then, it is said to kill the mutants otherwise mutant remains alive because it may be equivalent to the original program or the test cases taken are inadequate to find errors. If the mutant program produces the same output for all test cases then, it can't be killed and called as an equivalent mutant.

It is used to test the quality of test suite by killing the mutants and if is not able to kill the mutants then the test suites are inadequate. A test set which kills the entire nonequivalent mutant is said to be adequate.

2.1. Mutation Score

The Mutation score is calculated by dividing total number of nonequivalent mutant from total number of killed mutants [8, Mutation score always lies between 0 to 1. If mutation score is 1 then it implies the 100% adequacy of test cases.

$$\boldsymbol{mutation\ score} = \frac{\boldsymbol{killed\ mutant}}{\boldsymbol{total\ no.\ of\ mutants - equivalent\ mutant}}$$

An interesting case arises when a test set does not distinguish any mutant and all mutants generated are equivalent to the original program. In this case, distinguished mutant (killed) and live mutant is equal to 0 and the mutation score is undefined. It does not mean that test set is inadequate. In fact, the set of mutants generated are insufficient to assess the adequacy of the test set.

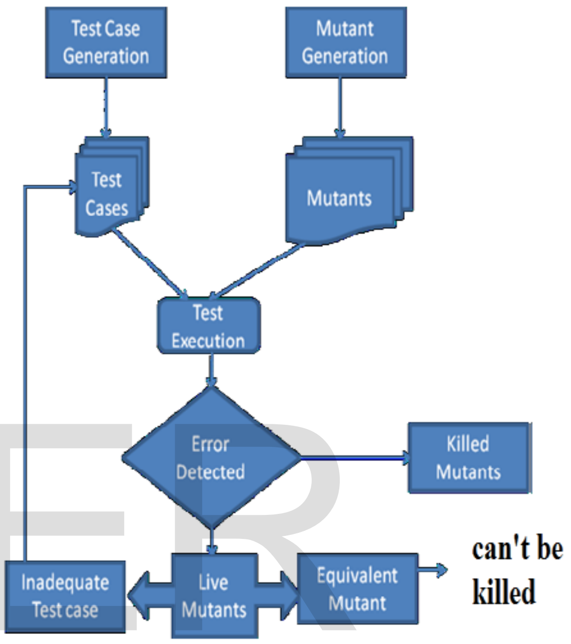

Figure 1: Mutation Testing Process [5]

2.2. Effectiveness of Test Cases

Test cases are effective by calculating the relation between Effectiveness of test case (E), mutation score and average number of test cases [5]-

$$\boldsymbol{E} = \frac{\boldsymbol{mutation\ score * average\ no.of\ testcase}}{\boldsymbol{Total\ test\ cases}}$$

Average numbers of test cases are calculated by dividing killed mutants from total number of dead mutants.

$$\boldsymbol{Average\ no.of\ test\ case} = \frac{\boldsymbol{Killed\ mutants}}{\boldsymbol{No.of\ dead\ mutants}}$$

2.3. Types of Mutants

A mutant is classified into one of the three types: error revealing, error hinting and reliability indicating [1].

2.3.1 Error Revealing:- A mutant M is said to be *error revealing* for program P if there exists at least one test case for which program P(t) is not equal to the mutated program M(t).

2.3.2 Error Hinting:- A mutant M is said to be *error hinting* if program P is equivalent to mutated program M but does not equivalent to the correct version of program P_c

2.3.3 Reliability Indicating:- A mutant M is said to be as *reliability indicating* if program P (t) is not equal to mutated program M (t) for some test cases but corrected version of program P_c is equal to program P.

3. Related Work

A paper on "An empirical based mutation testing through effective test data" has been presented for object oriented software [6]. Here classical approach was used for object oriented programming and specified the effectiveness of mutation testing in it

A.Ramya *et al.* has been provided an overview of mutation operators that plays an important role in mutation testing and also about the tools that help in automating the process in various languages has been specified [4].

The mutation testing process with cost reduction techniques were explained in the review paper on mutation testing [5].The process of mutation testing was discussed for finding faults in the mutant program.

A cost reduction technique has been proposed and the complexity of mutation testing has been reduced by using the concept of meta-data versioning [7]. Here metadata version tables were created which keep track of changed values, previous values before that changed, and timestamp of the transaction. This study presented both the advantages of providing the highest level of mutation coverage and reduction in space complexity.

A paper "A manifesto for higher order mutant" has been presented in which single higher order mutant was created to perform the testing process instead of more than one first order mutant to reduce the time and space complexity of mutation testing [8].

4. Equivalent Mutants

Equivalent mutants are those mutants which results in same output for every test case due to many reasons such as there may be same semantic with different syntax of program, the test cases never reach to the mutated statement, and error can't propagate to the output statement where the results was taken etc.

Given a mutant M of program P, then M is equivalent to if P (t) = M (t) for all possible test inputs t. That means, if M and P behave identically on all possible inputs, then two are equivalent[1].

Program(P)	Equivalent Mutant(M)

For(int i=10;i>5;i--)
{
Fprintf('Value of I is %d',i);
}
	For(int i=10;i!=5;i--)
	{
	Fprintf('Value of I is %d',i);
	}

Here, In the diagram, an equivalent mutant is generated by changing the greator than(>)operator into not equal to(!=) operator. If the statement within the loop doesn't change the value of i, then the program (P) and mutant (M) will produce the same output.

The general problem of determining whether a mutant is equivalent to its parent is not decidable and equivalent to the halting problem. Hence, in most practical situations, detection of equivalent mutants in mutation testing is done by the tester or by using automated testing tools through careful analysis [9]. And it is an interesting topic of research for automated detection of equivalent mutant.

5. Techniques to Reduce Mutants

As mutation testing is very time-consuming task, so some techniques must be required to reduce the effort and time of mutation testing process. One of the technique is the reduction in number of mutants in which the subset of mutants M' will be find such that subset of mutants MS_T(M') is equivalent(=~) to total number of mutants MS_T(M). There are four techniques available to reduce the mutants as:-

5.1. Mutant Sampling*:* It is the simplest approach in which a small subset of mutants will be chosen randomly from the entire set of mutants. In this approach y% of mutants are selected randomly from the total number of mutants and remaining are discarded. It was first performed by Budd [2].

5.2. Selective Mutant*:* In this approach, number of mutants can be reduced by reducing the number of mutant operators used. This is the basic idea which finds the subset of mutated operators from the entire set of mutants without significant loss of test effectiveness. It is more superior than mutant sampling and was first proposed by Mathur [1]. Offut [10] extended the work by omitting four and six selective mutation operators. Mutation operators are divided into three categories: statement, operands, and expressions [1]. Namin [9] performed his research work on selective mutant problem by formulating it into statistical problem..

5.3. Mutant Clustering*:* It was proposed by the Hussain [11] .Instead of selecting mutants randomly, subset of mutants will be chosen using some clustering algorithm. Different clusters will be created based on the killable test cases and each mutant that lies on the same cluster must be killable by a similar set of test cases. In mutant clustering, a small subset of mutants is chosen from each cluster for mutation testing process and remaining are discarded.

5.4. Higher Order Mutation (HOM): Higher order mutants are those mutants that contain more than one change in the program. Here, rare but valuable and less complex higher order mutant must be find that denote a subtle fault. Harman introduced the concept of HOMs [12]. HOM is sometimes harder to kill because it results in very complex structure of the program so it is always preferable to handle it carefully and less preferable than other techniques.

6. Conclusion

This paper introduces an overview on mutation testing and the process of mutation testing to distinguish between live mutants and kill mutants based on various test cases. On the basis of dead mutants and equivalent mutants, mutation score is calculated which specify the effectiveness of mutation testing by calculating the relationship between test cases and mutation score. After that various types of mutant are specified based on the various conditions. Mutation testing is an expensive method due to high effort and cost involved. Various techniques are described in the last phase to reduce the number of mutants which require lots of effort to find the subset of mutants.
The future scope of mutation testing will be the reduction in cost and time consumption by fully automating the testing process and also by reducing the equivalent mutant problem.

BIBLIOGRAPHY

[1] A. P. Mathur, Foundation of software testing, Pearson publication, 2009.

[2] D. A. T.A Budd, "Two Notions of correctness and their relation to testing," *Acta Informatica*, pp. 31-45, march 1982.

[3] .. h. R, "Testing programs with the AID of a Compiler," *IEEE Transactions on software engineering*, 1977.

[4] S. P. A Ramya, "An oversight on mutation testing," *International journal of engineering Research & technology*, jan 2014.

[5] P. K. Chaurasia, "Mutation testing:A Review," *Journal of global research on computer science*, Feb 2014.

[6] D. S. Prasad, "An emipirical based mutation testing through effective test data," *International journal of advanced research*, may 2015.

[7] r. B. R. M. R. A. Anu saini, "Reducing the cost and complexity of mutation testing using metadata versioning," *IJERT,* may 2013.

[8] M. .. H. Y.Jia, "A Manifesto for higher order mutation testing," CREST center King's College, London, 2009.

[9] J. A. A.S. Namin, "Finding sufficient mutation operators via variable reduction," in *Proceedings of the 2nd workshop on mutation analysis* , Raleigh, North carollna, Nov 2006.

[10] A. offut, "Investigations of the software testing coupling Effect," *ACM transaction on software engineering methodology,* jan 1992.

[11] S. Hussain, "Mutation Clustering," Master's Thesis,King's College, London, 2008.

[12] M. h. Y.Jia, "Constructing subtle faults using higher order mutation testing," in *International working conference on source code analysis and manipulation,* Beijing,China, sept 2008.