International Journal of Engineering & Technology, 7 (1.1) (2018) 243-245

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research paper

A study on mutation testing of object oriented programs

T. N. S. Poojitha *, K. V. L. Pushpanjali, D. Goutham, K. V. Yashwanth, Srinivas Prasad

Department of Computer Science and Engineering, Koneru Lakshmaiah Education Foundation
**Corresponding author E-mail: sivapoojitha15@gmail.com*

Abstract

Mutation testing is a modern approach which gives more appropriate results. In comparison to traditional approaches, it gives high quality output. Previously it is not used mostly because of its high cost factor. This is because mutation testing deals with white box testing. White Box testing checks every module of the software in detail. If we use this it takes a lot of time and money. Recent approaches which came in mutation testing made it easy to implement for any software. Mutation take a look ating could be a fault based mostly testing technique within which mutants area unit generated within the program and apply totally different test cases on the mutants. Some mutants are killed and some are alive. On the bottom of killed and alive mutants, mutant score is calculated. Based on the mutants which are alive the test cases can be improvised there by the quality of the source code is increased. we propose a tool which gives more effective output of testing. We propose a tool which takes the outputs of various static tools available and combines it with the outputs of dynamic tools available. Our proposed tool includes outputs of available tools like Jester, Mujava, PMD to effectively detect the vulnerabilities and produce high quality software as output.

Keywords: *Mutation Testing; Equivalent Mutants; Mutation Operators; Mutation Score.*

1. Introduction

In software testing Test coverage is the essential factor. Mutation testing helps in analysing the program if a group of testing techniques are sufficient to ensure that the product meets all the quality guidelines. If we are unable to find the ambiguities or errors, we can't gurantee that the system is free from errors. Mutation testing is an efficient way of fault based testing from the perspective of errors caused by programmers to improve the quality of test suite. Mutation Testing is a strategy that has been created utilizing two essential thoughts Competent Programmer Hypothesis reveals that the developers write programs that are somewhat contrasted from the coveted program and Coupling Effect Hypothesis reveals that recognizing basic errors will prompt the recognition of more severe mistakes that was initially proposed in 1970s by De Millo et al and Hamlet [1]. This method involves creation of mutants, which are small parts of source code modified to create suitable test cases. Mutants are detected and killed by making the original source code to differ from the mutant. The mutant score is calculated based on the number of mutants alive and the number of mutants killed. Mutation operators depends upon programming languages there are different open source tools to perform the mutation testing depending on the different programming languages, but there are some traditional mutation operators like deleting a statement, replacing the Boolean expressions, replacing of arithmetic operators, replacing the value or name of the variable. The best part of the mutant generation is that the mutation operators can be depicted absolutely and will provide a fault-seeding process [2]. Various mutation operators are decided based on the original program creating an indefinite set of mutant programs. We can change certain values of constants or attributes in the mutant program these are called as value mutations. We can change decision making operators in the source program to obtain the mutant program this type of mutants are called as decision mutations. We can delete a line of code in the source program or swap the lines of code to generate a mutant this type of mutants called as statement mutations.

2. Process of mutation testing

Mutation Testing is a structural testing approach that can be used to check the efficiency or the precision of a particular software or program. There is a characterized procedure to implement the mutation testing that is as per the following, contemplating the actual code on which mutation testing has to be implemented. Presently errors are brought into the actual code by making numerous renditions called mutants. There can be on one fault for each mutant so we require more number of mutants for the same source code and the goal of this mutation testing is to fail the mutation version of code which confirms the efficiency of the test case. We build certain test cases for our source code which are together termed as the test suite now after introducing the faults to the source program this program is termed as the mutant program. we can find the test case adequacy by applying the test cases to the actual code and to the mutant code. On contrasting the first program with that of mutant program if the first program and the mutant program produce a similar yield then that mutant is slaughtered by the experiment. From this we can affirm that the experiment is sufficient to distinguish the change amongst unique and the mutant program. In the event that the yield produced by the source program and mutant program is diverse then that mutant is kept alive all things considered we have to enhance the test cases in order to execute every one of the mutants. Mutant score is computed to calculate the sufficiency of the test cases the proportion of number of mutants slaughtered to that of aggregate number of mutants duplicated by hundred results in mutation score. Mutation testing suffers from equivalent mutants. Equivalent mutant acts in the same behaviour of the source program. On introducing a change to the source program does not modify the meaning of the original program.

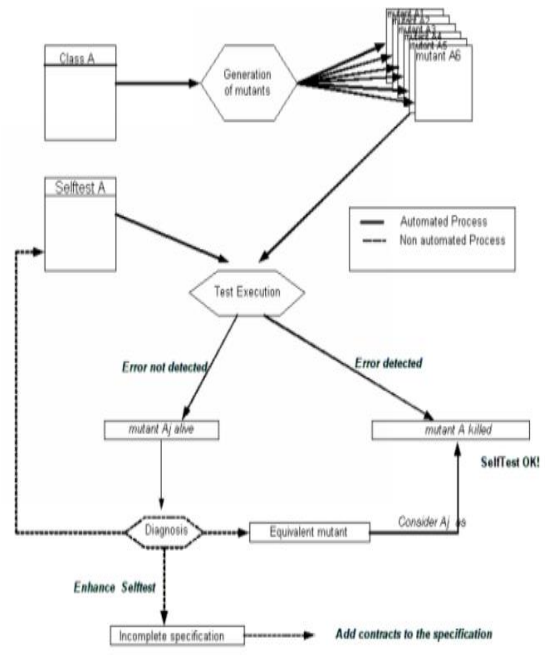

Fig. 1: Process of Mutation Testing.

3. Types of mutation testing

Mutation testing can be carried out in three ways, they are as follows. Strong mutation testing: A mutation testing is called as strong only if it fulfils the accompanying conditions. (1) the test must reach the statement that is mutated. (2) the input data that is to be tested should be infected by the program state by enabling varying program states for the mutant program and the original source code. Consider the following example, a test x=1 and y=0 will accept this condition. (3) the program state that is incorrect must propagate to the program output and the test is checked. The above three conditions are together called as the RIP model. Weak mutation testing: A mutation testing is called as weak mutation if these two conditions are satisfied they are (1) Mutated statement must be reached by a test. (2) the input data that is to be tested should infect the program state by causing different program states for the mutant program and the original program. Consider the following example, a test x=1 and y=0 will accept this condition. Weak mutation testing requires less computing power when compared to that of strong mutation testing. Weak mutation testing is firmly related to the coverage methods. Firm mutation testing: the mutation testing which falls in the middle of both the strong and weak mutation testing is called as firm mutation testing. For firm mutation, it is not expected that an assertion catches the difference in behaviour. unlike weak mutation, where the change that is induced should propagate some distance from the place of the origin.

4. Mutation operaotrs

Mutation operators defines the type of operation that has to be changed in the source program to generate the mutant program. There are two types of sorts they are method level mutation operators and class level mutation operators. (1) MuJava changes the expressions by insertion, replacement or deletion of the primitive operator. Method level mutation can be performed on six types of operators [7]. (2) Class level mutation operators are implemented in MuJava. There are four categories of mutation operators they are broadly classified as follows: Encapsulation, Inheritance, Polymorphism, Java-specific Features .Encapsulation:

(1.1)Encapsulation oversees data stowing endlessly. It is the limit of an inquiry make a point of constrainment around its data and techniques. Encapsulation empowers an engineer to describe the passageway particular articles. For this numerous access modifiers are used. Deciding the wrong access modifier can incite erroneous outcomes. We use a passageway modifier change overseer to change the passage modifier of the source program. This empowers an analyzer to ensure that the correct level of accessibility is used as a piece of a program.
(1.2)Inheritance: The data that is present in the once class can be inherited or used in the other class this is called inheritance. Code reusability is the special feature that is present in the inheritance.
(1.3)Polymorphism: Polymorphism enables items to respond distinctively to a similar strategy. It is actualized by having numerous strategies with a similar name.

5. Advantages of mutation testing

Mutation testing is an intense way to deal with achieve high scope of the source program. We can build the extent of testing this aides in accomplishing higher norms. In regular testing methods the scope of the testing is only limited to some pre conditions but in mutation testing the scope is widely extended. The end users or the customers are highly benefited by mutation testing as a system that undergoes the mutation testing is highly reliable and stable to that of a system that does not include mutation testing. Mutation testing has the capability to uncover all the uncertainties that are in the source code which cannot be done by almost all other testing methods. It can be applied parallel to other testing methods so as to get higher efficiency of the source program. Mutation testing is a powerful mechanism to detect the testing inadequacies or to check the coverage on testing of the particular source code by using the mutant program. The steps that are involved in the mutation testing are fully automated such as creating the mutants, mutation operators, results which reduce the human effort. We can perform the mutation testing manually or using an automated tool so the developer has the choice to select the testing that best suits the project. It can distinguish the undetectable deformities that can't be recognized by the consistent testing systems there are sure imperfections that can't be recognized by the other testing strategies however those imperfections can be recognized by utilizing the change testing as the extent of the transformation testing is high contrasted with that of other testing techniques.

6. Disadvantages of mutation testing

Mutation testing is extremely costly as we need to generate a mutant program for the original program. Mutation testing involves source code changes so this method is not suitable to the black box testing. Each change program has a similar number of experiments to that of the first program along these lines, more number of mutant projects should be tried. Mutation testing is time consuming as it requires to generate mutant program for the original program if the source program is large then more number of mutant programs need to be generated each has some finite number of test cases which takes more time. As it takes more time it cannot be tested manually we need an automation tool for that purpose. This mutation testing is not user friendly as we must understand the complete features of the automation tool which takes more time and requires human effort. Transformation testing requires many experiments to recognize the mutant from the first source code. Change testing is hard to actualize on account of complex transformations or complex projects. Mutation testing isn't appropriate to the equal mutants as the computerization instrument can't recognize the blunders in comparable mutants so that the part of the original program that contains the equivalent mutants must be identified and the testing has to take place manually on these equivalent mutants.

7. Tools review

Mutation testing can be done in many programming languages depending on the convenience of the software tester following are the automated mutation testing tools based on the programming language.

7.1. Jester

Jester is an open source tool used for the mutation testing and jester can also be used as the additional plugin for the eclipse IDE. Jester is a motorized change testing device that is used to test the java programs. Jester works with JUnit tests. Jester does elementary adjustments to the projects, for example, changing If proclamations to genuine or false, and so forth. Subsequent to making these alterations, it runs tests on the adjusted projects. a inherent script is used to generate the webpages to show the results. Jester is entirely distinct to that of code coverage tools. jester's approach is called as the machine-controlled error seeding. we can't consider jester as the substitute for the code coverage tools it is a reciprocal approach [5].

7.2. Jumble

Is a basic non-realistic open source robotization instrument for change testing. It changes over the substance reports into interpretation that enables perusing the organization of the record. muddle works straight forwardly at a source code level and quicken the testing process. The compelled number of Mutation administrators maintained by Jumble are according to the accompanying: increases, Conditional, switch articulations, Binary Arithmetic Operations, Return Values, Inline Constants and Class Pool Constants. Disorder could be a class level change testing instrument that works alongside JUnit. A transformation is performed on the source code that must be tried. On the off chance that there is a mistake created amid the execution of the change program then the experiments are sufficiently productive to identify the blunders. Alternately if the change program does not demonstrate any mistake on execution of the transformation program then the experiments are not sufficiently proficient to identify the blunders for this situation we need to enhance the experiments.

7.3. µJava (mujava)

MuJava is a computerization instrument to perform change testing on java programs. Mujava utilizes two gatherings of change mutation operators they are technique level and class-level. MuJava utilizes numerous technique level and class-level mutation operators to make the mutant projects. At that point the experiment are executed on the mutant projects and assesses the transformation scope on the mutant projects. Transformation mutation operators considers the program under test and roll out the important syntactic improvements on it. These syntactic changes portray regular linguistic slip-ups made by software engineers while composing code. MuJava realizes a 'do faster' approach to manage change testing to save collection time [6]. This 'do faster; approach is best sensible for challenge arranged undertakings. The arrangement of MuJava utilizes the Mutant Schemata Generation approach.
Following are the two courses of action that are used by the mutant schemata age approach they are aggregation of the main program and assembling of the meta-mutant program. The change transformation administrators that are used by the Mujava for the change testing are of two sorts they are structure change administrators and lead change transformation administrators. The Mujava gadget make the structure and direct mutants. For the lead mutants, orchestrate time reflection is utilized to separate the fundamental program. The MSG engine at that point employments gather time reflection likewise to make a meta-mutant program. For the fundamental mutants, the primary source code is accumulated using the Java compiler. BCEL API is then used to incorporate or delete class people in the subsequent byte code interpretation [6].

8. Conclusion

Mutation testing is a form of white box testing which requires the change in the source code. Mutation testing methodically assesses the nature of existing test cases by calculating the mutation score. No matter how, mutation testing suffers from equivalent mutants in which the testing has to be carried out manually, and a high computational cost related with a large pool of generated mutants for the original program. Mutation testing can be applied to all the design phases, coding. Apart from testing phase it can also be applied to other phases of the project. Mutation testing can be implemented parallel to the testing phase to achieve high quality test cases and quality software meeting all the standards. Using mutation testing teams get higher performance related to quality test cases.

References

[1] Problems of Mutation Testing and Higher OrderMutation TestingQuang Vu Nguyen, Lech MadeyskiInstitute of Informatics, Wroclaw University of Technology, WybrzezeWyspianskiego 27,50370 Wroclaw, Poland.

[2] Is Mutation an Appropriate Tool for Testing Experiments? J.H. Andrews Computer Science Department University of Western Ontario London, Canada and L.C. Briand Y. Labiche Software Quality Engineering Laboratory Systems and Computer Engineering Department.

[3] A Comparative analysis of Mutation Testing tools for Java Forostyanova Mariya PhD Student at National Research Tomsk State University mariafors@mail.ru Dongak Barkhas Student at National Research Tomsk State University.

[4] Mutation Operators for Concurrent Java (J2SE 5.0)1 Jeremy S. Bradbury, James R. Cordy, Juergen Dingel School of Computing, Queen's University Kingston, Ontario, Canada {bradbury, cordy, dingel}@cs.queensu.ca

[5] Moore, I. "Jester – A JUnit Test Tester". Proceedings of the 2nd International Conference on Extreme Programming and Flexible Processes in Software Engineering (XP2001).

[6] Ma, Y. S., Offutt, J. & Kwon, Y. R. "MuJava: An Automated Class Mutation System". Journal of Software Testing, Verification and Reliability, 15(2):97-133, June 2005. https://doi.org/10.1002/stvr.308.