International Journal of Science and Research (IJSR)
ISSN: 2319-7064
SJIF (2019): 7.583

Mutation Testing Techniques in Software Testing: A Review

Dushyant Singh[1], Parulpreet Singh[2]

[1]Research Scholar, Baddi University of Emerging Sciences and Technology, Baddi, India

[2]Assistant Professor, Baddi University of Emerging Sciences and Technology, Baddi, India

Abstract: ***Mutation testing is regarded as a dominant scheme to quantify the quality of test suite. There are two stages for executing mutation testing. The initial stage includes the alteration of code into various instances known as mutants. The compilation of these mutants is done later on. A mutation engine or manual way is utilized to produce and compile the mutation in automatic manner. The various schemes which are based on the mutation testing are reviewed in this paper.***

Keywords: Mutation Testing, Genetic Algorithm, Manual Testing

1. Introduction

Software testing is a fundamental way to assess the qualities of software products. The testing of software must be done carefully. However, testing is very difficult task and error-prone activity and it is not considered more amusing than creative activities like programming. It is seized for practitioners and in higher education at which testing is not much popular. The reason behind less adoption of latest testing schemes including mutation testing is that testing has not attained much attention regardless of having a capability of enhancing efficiency to detect the faults of testing activities. Mutation testing is very challenging to be executed for dealing with computational problems and its application in practice [1]. The software faults are simulated in a program through MT for computing the potential of state-of art test suite for detecting fault. Minor changes are carried out in the original code in systematic manner for generating various modified versions. Every modified version is known as a mutant and it is in correspondence with the application of one particular mutation operator that is pre-defined. A mutation operator is assisted in categorizing the change's kind that is adaptable for a program in impersonating typical syntactic errors programmers make.

Generally, mutation testing is regarded as a dominant scheme to quantify the quality of test suite. There are two stages for executing mutation testing. The initial stage includes the alteration of code into various instances known as mutants. The compilation of these mutants is done later on. A mutation engine or manual way is utilized to produce and compile the mutation in automatic manner. Every mutant is a replica of original problem excluding one atomic change. The atomic change is made based upon a specification embodied in a mutation operator [2]. The atomic change is accomplished on the basis of the specification embodied in a mutation operator. There are two main assumptions on which execution of atomic changes under mutation testing is relied. It is defined in the Competent Programmer Hypothesis that developers are often liable for developing a program that is close to be correct. In coupling effect, it is assumed that test cases distinguishing programs with slight changes are so sensitive that may differentiate programs with further complicated differences. The language constructs generated to alter classify mutation. In the past, the range of operators was restricted to statements within the structure of a solo procedure. This type of operators is generally termed as traditional, or method-level, mutants. For instance, a single traditional mutation operator alters1 binary operator (e.g. &&) to another (e.g. ||) in a try to generate a fault variant of the program. Nowadays, operators that test at the object level, or class-level operators have been devised. For example, some specific class-level operators in the Java programming language substitute method calls within source code with a same call to a dissimilar method. Class-level operators make use of the object-oriented traits of a provided language [3]. The range of possible mutation for adding specifications for a given class and inter-class execution are expanded using these features. The second part of mutation testing executes a test suite against a mutant and records pass/fail results. The mutant is said to be killed when the test results of a mutant differ than the original's test result. This implies that at least one test to catch the mutations was satisfactory. The test results of a mutant similar to the original let the mutant to be alive or alive. This indicates that the transformation given by the mutant has survived the test cases. The mutants that are impossible to be killed because of logical equivalence with the original code or because of language constructs are called Stubborn mutants [4]. The number of killed mutants is divided by the total number of mutants to calculate a mutation score. A mutation score of 100% indicates about the adequacy of the test suite. However, a mutation score of 100% may not be achieved due to the inevitability of stubborn mutants. Practically, mutation analysis generates a test set that kills all mutants that can be killed (i.e., are not stubborn).Mutation testing is generally considered to be a costly testing method in regard to computation. But this consideration is partially relying on the obsolete hypothesis that each mutant in the traditional Mothra set must be taken into account. A number of cost reduction methods have been presented by so far to turn Mutation Testing into a real-time testing method. Cost reduction techniques are generally divided into two types, namely Mutant Reduction Techniques, Execution Cost Reduction Techniques. A main source of computational cost in Mutation Testing involves the inherent running cost in

executing many mutants against the test set [5]. Due to this, reduction in the number of created mutants without noteworthy loss of test efficiency has turned out to be a hot research issue. Mutant Sampling, and Mutant Clustering are the two most popular approaches, that are used for reducing the number of mutants. Mutant Sampling is a easy to use method. In this method, a small subset of mutants is selected randomly from the overall set. This approach firstly generates all possible mutants just like in traditional Mutation Testing. Afterwards, this approach randomly selects x% of these mutants for mutation analysis, while rejects the rest. In Mutant Clustering, a subset of mutants is selected by clustering algorithms rather than choosing mutants randomly. Mutation Clustering initially generates all mutants of first order. Subsequently, the first order mutants are classified into different clusters depending on the killable test cases by applying a clustering algorithm [6]. A similar set of test suites guarantees the killing of every mutant in the similar cluster. Merely few mutants are chosen from every cluster to be employed in Mutation Testing, the rest mutants are rejected. Execution Cost Reduction Techniques not only reduces the number of created mutants, but also reduces the computational cost by making the mutant execution process optimized. Run-time Optimization Techniques represents an Interpreter-Based Technique. This technique is employed in the first generation of Mutation Testing tools [7].The most commonly used technique for achieving program mutation is Compiler-Based Technique. This technique starts by compiling every mutant into an executable program. After this, a number of test cases are used for executing the complied mutant. In comparison to source code interpretation methods, this approach is too fast as compiled binary code is executed in less time than interpretation. Nevertheless, a speed limitation called compilation bottleneck occurs because of the high compilation cost for programs whose execution time is much greater as compared to the compilation/link time.

2. Literature Review

Nishtha Jatana, et.al (2020) suggested PSO-MT to generate the test data [8]. This approach was implemented on larger programs from SIR in order to strengthen it. Same working qualities were found from Particle Swarm Optimization as those of GA that was employed to compute the test data with mutation testing. The comparative analysis of suggested approach was done with GA-MT to generate the test data. The outcomes demonstrated that a superior efficiency had obtained using the suggested approach together with Mutation Testing for most of the benchmark programs in comparison with the GA-MT. The obtained outcomes were analyzed in statistical manner by carrying out Statistical test.
Pablo C. Cañizares, et.al (2018) intended a mutation testing model to detect the errors in distributed applications whose deployment was done in simulated environments [9].The test suite was implemented against the set of mutated models for determining the efficiency of intended model to detect diverse errors. Mu Tom Vo was employed to exploit this proposal. There were 3 applications of different distributed systems had utilized to perform a case study so that viability of the proposal was sustained.

Yunqi Du, et.al (2019)a mutation operator selection strategy was designed on the basis of Selective Mutation for alleviating the number of mutants [10]. There were five mutation operators chosen out of 19 operators of Mujava for attaining a subset. This subset was employed in test cases that provided the average variation score above 95% on the variants of the whole set. Afterward, a test case generation technique was recommended in which MT was integrated with GA. This assisted in redefining various operators of test cases and optimizing those test cases. At last, a set of test cases that included better coverage and greater mutation score had acquired after the comparison of recommended strategy with some algorithms and tools.

Lingchao Chen, et.al (2018) analyzed that the MT was the effective scheme for computing the quality of test suites [11].Regression Testing Selection tools were employed for boosting mutation testing of advanced software systems. Thus, first extensive study was carried out in order to evaluate the efficacy and efficiency of several Regression Testing Selection schemes that had utilized for the maximization of mutation testing. The outcomes of study exhibited that file-level static and dynamic RTS were capable of providing mutation testing accurately and effectively and they also offered practical guidelines for developers.

Do Van Nho, et.al (2019) described that Mutation testing in general and higher order mutation were methods employed for computing the quality of test data which classified that the test data was capable of uncovering the errors or not [12]. But, higher order mutation was quite expensive due to the enormous number of produced mutants. The main purpose of this work was to mitigate the cost of higher order MT. Several strategies were suggested for incorporating mutants of first order so that less number of higher order mutants had generated for a program under testing preserving the quality of generated mutants. A set of different programs were utilized to experiment the presented strategies. The outcomes validated that the presented method with respect to the number of mutants and mutation score which had produced.

Francisco Gomes de Oliveira Neto, et.al (2020) stated that the mutation testing was carried out for quantifying the behavioural diversity [13]. In order to achieve this, the set of test cases was executed on different mutated versions of the SUT. There were 2 specific b-div measures introduced and their comparison was done with a-div to prioritize the test suites included in six different open-source projects. The outcomes represented that the b-div measures performed better as compared to a-div and random selection in all of the studied projects. The APFD was enhanced from19% to 31% on the basis of subset's size of prioritized tests.

Farah Hariri, et.al (2019) emphasized on employing a MT tool known as SRCIROR in which the same mutation operators were exploited at both levels [14].The automated methods were utilized for justifying equivalent and replicated mutants and also for classifying the mutants as minimal and surface. The study was conducted using fifteen programs taken from the Core utils library. It was evaluated that the MT was superior at SRC level as it was not costly

and generated less mutants. A case study was also conducted using Space program for computing the level at which mutation score correlated with the actual capability of detecting faults of test suites sampled was higher. It was revealed that the mutation score at both levels was not much correlated with the actual fault-detection capability of test suites.

Author	Year	Description	Outcomes
Nishtha Jatana	2020	This approach was implemented on larger programs from SIR in order to strengthen it. Same working qualities were found from Particle Swarm Optimization as those of GA that was employed to compute the test data with mutation testing.	The outcomes demonstrated that a superior efficiency had obtained using the suggested approach together with Mutation Testing for most of the benchmark programs in comparison with the GA-MT.
Pablo C. Cañizares	2018	The test suite was implemented against the set of mutated models for determining the efficiency of intended model to detect diverse errors. Mu Tom Vo was employed to exploit this proposal.	There were 3 applications of different distributed systems had utilized to perform a case study so that viability of the proposal was sustained.
Yunqi Du, et.al	2019	Regression Testing Selection tools were employed for boosting mutation testing of advanced software systems. Thus, first extensive study was carried out in order to evaluate the efficacy and efficiency of several Regression Testing Selection	At last, a set of test cases that included better coverage and greater mutation score had acquired after the comparison of recommended strategy with some algorithms and tools.
Lingchao Chen	2018	The MT was the effective scheme for computing the quality of test suites. Regression Testing Selection tools were employed for boosting mutation testing of advanced software systems.	The outcomes of study exhibited that file-level static and dynamic RTS were capable of providing mutation testing accurately and effectively and they also offered practical guidelines for developers
Do Van Nho	2019	Mutation testing in general and higher order mutation were methods employed for computing the quality of test data which classified that the test data was capable of uncovering the errors or not	The outcomes validated that the presented method with respect to the number of mutants and mutation score which had produced.
Francisco Gomes	2020	In order to achieve this, the set of test cases was executed on different mutated versions of the SUT. There were 2 specific b-div measures introduced and their comparison was done with a-div to prioritize the test suites included in six different open-source projects.	The outcomes represented that the b-div measures performed better as compared to a-div and random selection in all of the studied projects. The APFD was enhanced from19% to 31% on the basis of subset's size of prioritized tests.
Farah Hariri	2019	The automated methods were utilized for justifying equivalent and replicated mutants and also for classifying the mutants as minimal and surface. The study was conducted using fifteen programs taken from the Core utils library.	It was revealed that the mutation score at both levels was not much correlated with the actual fault-detection capability of test suites.

3. Conclusion

In this work, it is concluded that mutations testing is the efficient testing scheme which detect errors efficient from the software's. The various mutations testing schemes are reviewed in this paper which for the defect prediction. It is analyzed that schemes which are based on the genetic algorithm are efficient for the mutation testing.

References

[1] Zainab Nayyar, Nazish Rafique, Nousheen Hashmi, Nadia Rashid, Saba Awan, "Analyzing test case quality with mutation testing approach", 2015, Science and Information Conference (SAI)
[2] Vasundhara Bhatia, Abhishek Singhal, "Design of a Fuzzy model to detect equivalent mutants for weak and strong mutation testing", 2016, International Conference on Information Technology (InCITe) - The Next Generation IT Summit on the Theme - Internet of Things: Connect your Worlds
[3] Tomohiko Takagi, Takuya Arao, "Overview of a place/transition net-based mutation testing framework to obtain test cases effective for concurrent software", 2015, IEEE/ACIS 16th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)
[4] Liping Li, Honghao Gao, "Test suite reduction for mutation testing based on formal concept analysis", 2015, IEEE/ACIS 16th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)
[5] Pedro Reales, Macario Polo, José Luis Fernández-Alemán, Ambrosio Toval, Mario Piattini, "Mutation Testing", 2014, IEEE Software, Volume: 31, Issue: 3
[6] Shabnam Mirshokraie, Ali Mesbah, Karthik Pattabiraman, "Guided Mutation Testing for JavaScript Web Applications", 2015, IEEE Transactions on Software Engineering, Volume: 41, Issue: 5
[7] Panya Boonyakulsrirung, TaratipSuwannasart, "A weak mutation testing framework for WS-BPEL", 2011, Eighth International Joint Conference on Computer Science and Software Engineering (JCSSE)
[8] NishthaJatana, Bharti Suri, "Particle Swarm and Genetic Algorithm applied to mutation testing for test data generation: A comparative evaluation", 2020, Journal of King Saud University - Computer and Information Sciences
[9] Pablo C. Cañizares, Alberto Núñez, Mercedes G. Merayo, "Mutomvo: Mutation testing framework for simulated cloud and HPC environments", 2018, Journal of Systems and Software
[10] Yunqi Du, Ya Pan, Haiyang Ao, NimakoOttinah Alexander, Yong Fan, "Automatic Test Case Generation and Optimization Based on Mutation Testing", 2019, IEEE 19th International Conference on

Software Quality, Reliability and Security Companion (QRS-C)
[11] Lingchao Chen, Lingming Zhang, "Speeding up Mutation Testing via Regression Test Selection: An Extensive Study", 2018, IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)
[12] Do Van Nho, Nguyen Quang Vu, Nguyen Thanh Binh, "A Solution For Improving The Effectiveness of Higher Order Mutation Testing", 2019, IEEE-RIVF International Conference on Computing and Communication Technologies (RIVF)
[13] Francisco Gomes de Oliveira Neto, Felix Dobslaw, Robert Feldt, "Using mutation testing to measure behavioural test diversity", 2020, IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)
[14] Farah Hariri, August Shi, Vimuth Fernando, Suleman Mahmood, Darko Marinov, "Comparing Mutation Testing at the Levels of Source Code and Compiler Intermediate Representation", 2019, 12th IEEE Conference on Software Testing, Validation and Verification (ICST)