International Journal of Advance Research in Science and Engineering
Volume No.06, Issue No. 12, December 2017
www.ijarse.com
IJARSE
ISSN: 2319-8354

SURVEY ON MUTATION TESTING COST REDUCTION METHODS

Mrs.R.Shobana[1], Dr.G.Maria Prisicilla [2]

[1]DEPARTMENT OF COMPUTER SCIENCE ,Ph.D RESEARCH SCHOLAR,SRCAS,(INDIA)
[2]DEPARTMENT OF COMPUTER SCIENCE ,PROF & HEAD,SRCAS,(INDIA)

ABSTRACT

Mutation testing is a potential but computation wisenot so economic testing strategy.Varioustechniques have been designed in order to minimize the expense of mutationtesting by reducing the number of mutants that have to be run; but, manyof these techniques are not as efficient compared to mutation testing that makes use of a completeset of mutants.Those technical works mention about the techniques and strategies, which can be utilized for finding the mutation techniques over the software development. Few of the worksdepict the varioustypes of optimization techniques for decreasing the expense of mutation testing and choose the optimized test cases. This research focuseson studyinga variety of research techniques, which are focused on discovering theevolutionary and mutation testing techniques, employed for avoiding the importantproblems and generate optimal solutions.The available research techniques are compared along with their pros and cons, such that the research works in the future can concentrate on them more. Also, the experimental tests were conducted all the technical works and then their comparison made against each other so as to get the better approach under different performance criteria.

Key terms: Mutation Testing, Object Oriented Programming, test cases, optimization algorithm

I.INTRODUCTION

Software has seen extensive distribution in the last few years. During the earlier stages of the process of software development, the software product's requirements and specifications are defined with the purpose of explaining the requireduse for the product. With the system developing, it mustoblige to these requirements. A software product's most essential requirement is that it should satisfy the customer requirements. In the present day environmentwith diversifying business, time-to-market acts as a key factor inproducing a successful product. The concerns of Software quality are quite vast, inclusive of correctness, robustness, comprehensibility and adaptability. Quality can be determined by means of the customer satisfaction with the resultant system built depending on the requirements, which are integrated with success in the system. Owing to the significance of the quality pertaining to the software product, greater than fifty percent of the expense of software development is dedicated to testing. Software testing is the primary technique used for softwarequality assurance[1].

The aim of software testing is involved with the process concerned with the verification and validation of software applications or program such that it attains the requirements and development design, as required by the project or user. The first mutation testing was introduced in 1970 and its tool was realized by the Timothy Budd in 1980 in his research work [2]. As per Budd "Mutation testing is a fault based testing method, which seeks the errors in the program and the errors are found". These faults get seeded in the actual program and the mutated program is compared. Mutants are brought in when the programs are executed by the mutant operators. Every mutation generates a mutant program, created by a mutation operator.

Mutation testing is a fault-based testing methodthatyields a testing criterion known as the "mutation adequacy score". The mutation adequacy score can be utilizedfor measuring the efficiency of a test set with regard to its capability of detecting faults. During the testing process, the test cases are chosen and then executed. Then the comparison of the results are done with the result that is estimated. Even with a good planning in place, it is impractical to have all the planned test cases executed, much within the time and budget for the complete number of test cases. Industrial collaborator mentioned that one of his products withnearly 20,000 lines of code, needed seven weeks to execute the whole set of test cases. During the time of regression testing the complete set test cases have to be re-executed, each and every time a modification is made to just a small part of the code.

Mutants can be implanted in any place in the source code, and mutation coverage equals to structural coverage. A higher order mutant is typically a program, which can be got by the application ofvarious operations from a set consisting of first order mutant operations. The association between simple faults and complex faults can be examined. Simplefaults are described to be the first order mutants whereas complex faults are generally higher order mutants. Higher order mutation Testing is involved with more than one mutation position present within a module or program. An application program has been developed in order to make the creation of mutants of single order automated along with the higher order making use of arithmetic and relational mutant operators.

Mutants are run with test data that is suitably designed and the mutation score is computed. Test cases are developed in order to have the coverage of the mutation position and probability of the output returned by the actual program. Mutation Score provides the measure of the sufficiency of test set.The efficiency of mutation testing is dependenthugely on the kinds of faults, which are represented by the mutation operators. Therefore, the quality of the mutation operators acts as the key to mutation testing.

Object-oriented programming has severalhelpful features, like information hiding, encapsulation, inheritance, polymorphism, and dynamic binding. Even though these object oriented features facilitate the developers to build the systems in a more systematic and practicalmanner, new sorts of faults are introduced [3]. In order to find these faults, researchers have suggestedtechniques, which use the mutation testing forobject oriented features.

With the purpose of applying mutation testing to object-oriented programs, researchers have made use of theavailable mutation operators that were designed for procedural-language programs, to object-oriented programs. Researchers have also designedextra mutation operators, known as class mutation operators, to find faults that are specific to object-oriented. But, very less research has been carriedout to exhibit the resourcefulness of the class mutation operators. For instance, no proof exists that the class mutation operators produce realistic faults or that they create a reasonable number of mutants.

In order to deal with these challenges, in [4] performed a set of empirical studies employing an object-oriented mutation system, MuJava. The goals of the studyinclude (1) to identify how many number of mutants and what types of mutants are created for object-oriented programs and (2) to examine the number of class mutation operators' model faults, which are not found with conventional mutation testing. Thecostof mutation testing is a most sophisticatedchallengethat is assessed and examined by different researchers employingdiversetechniques. In this research, those research techniques are explained in terms of their operating procedure and their functions in addition to theirdifferent performance criterion. The advantages and disadvantages observed in those techniquesare also discussed indetail.

II.RELATED WORK

In [5], Zhang et al (2009) introduced a technique using the relative reliability test and operation paths' reliability prediction in order to have the test allocation of software reliability test adjusted in object-oriented programming that, in turn, is dependent on the actual operational profile. In this new technique, software reliability test is not justdependent on the operational profiles but also directed by the relative reliability prediction results of the operation paths. The adjustable range is determined by the actual operation running rate and the operation independence factor that can be obtained from neural network learning algorithm and is differentamong different software.

In [6], Huang et al (2014) employedParticle Swarm Optimization (PSO) algorithm thatproposed the group self-activity feedback (SAF) operator and Gauss mutation (G) changing inertia weight to enhance the performance of particle swarm optimization (PSO). Making use of the enhanced algorithm in software test case, the experiments indicate that the introducing a single path fitness function structure and multi-path fitness computation of parallel thinking provides superior results as for the iteration time in single path test compared to the standard PSO and is more effective in the generation of multi-path test case.

In [7], Li et al (2015) suggested about reducing the expense of mutation testing,and presented an algorithm for mutation test generation, andthen providedfew reduction rules to minimize the set of testsuite that isemployed for killing mutants depending on formal concept analysis. In the case of mutation testing, few representative errors areintentionally seeded into the SUT (System under Testing)to generate a set of faulty programs known as mutants, and all current test cases are executed on all the mutants. Designing efficient andhelpful mutation operators are one among the key challengesof mutation testing.The resultsproved thatthistechnique can produce a smaller size test suitecompared to other techniques. Moreover,this technique can be of someassistance to mutation testing.

In [8], Gong et al (2015) introduced a dynamic mutation execution techniqueand the mutation-based fault localization scheme with thetechnique, referred to as Faster-Mutation-based Fault Localization (MBFL). The dynamic mutation executiontechniquecomprises of two optimizations, which are mutation executionoptimization and test cases execution optimization. Theseoptimizations are focused on quicker computing suspiciousness valuesof statements through the dynamic adjustment of the order of execution ofmutants and test cases.

In [9], Souza et al (2016) proposedanautomated test generation strategy, employing hill climbing, forpowerful mutation. It incrementally targets at killing the mutantsstrongly,by concentrating on the propagation of mutants', i.e., the means of killing themutants, which are killed weakly but not strong enough. Moreover,the empirical results concerned with the cost and efficiency of this approach over a set of 18Cprograms. Thistechniqueattained a higher mutationscore compared to random testing, by 19,02% on an average, and theearlierintroduced test generation techniques, which neglect the propagation ofmutants', by 7,2% on an average. The results alsoindicate the improvement ofthis techniqueover the earlierstrategies.

In [10], Prajapati et al (2016)presented animproved data flow based Quality Assurance (QA) model for Component-basedSoftware Engineering (CBSE) by using the Ant Colony Optimization (ACO)algorithm for optimizing the code given for automatic generation and prioritization of optimal path in decision to decision Control Flow Graph (CFG) that, in turn, leads to an improved testing phase for QA model with minimized complexity. Then,the new ACO based technique is also used for generating the test data to meet the created set of paths. The results indicate that better testing is accomplished by using thenew ACO based strategy on component based software. Thistechniqueguaranteescomplete software coverage with least amount of redundancy.

In [11], Ma et al (2016) introduces a novelscheme for the execution oflesser mutants when retaining just the same level of efficiency as generated by mutation testing employing a complete set of mutants. Thistechnique performs the dynamic clustering of expression-level weakly killed mutants, which are anticipated to generate the same result under a test case; just one mutant from every cluster is completely executed under the test case. Thistechnique was implemented and its demonstration showed that it efficiently minimized the expense of mutation testingwith no loss in the effectiveness.

In [12], Bashir et al (2017) showed an enhanced genetic algorithm, which can minimize the computational expense of mutation testing. At first,it introduces a new state-based and control-specific fitness function, which makes efficient use of object-oriented program features for the evaluation of a test case. After this, itdoes the empiricalevaluation of it employingthis toolimplemented, eMuJava, and then performs itscomparison with standard fitness function. Results indicate that even though thenew fitness function yields detailed information regarding the fitness of a test case but the standard genetic algorithm is quite not capable of making use of that with efficiency in order to correct the test cases. Therefore, new two-way crossover and adaptable mutation techniques, which intelligently utilize the fitness information for generating a fitter offspring is proposed. At last the enhanced genetic algorithm with random testing, standard genetic algorithm, and EvoSuite are compared. The experimental results illustrate that this new technique can detect the optimal test cases in lesser attempts (minimizes the computational expense). Also, it can find the software bugs from doubtfully equivalent mutants and these mutants are eventually killed (maximizes mutation score).

III.EXPERIMENTAL RESULT

An experiment is conducted, where a comparison is made withEvoSuite [30], the tool having the most relevance in the literature. EvoSuite can be used for testing the Java based programs and can produce tests for branch coverage and mutation based coverage. It just provides support to unit level testing such thatEvoSuite cannot use mutation operators, which involves more than one class (mutation operators for inheritance or polymorphism). Moreover, it does not provide support to mutation operators, which change simple objects (access modifier related mutation operator and object vs. reference comparison etc). Generally, EvoSuite does not provide support to any object-oriented feature and makes use ofrestricted set of classical mutation operators for the generation of mutants. Owing to this reason, even though EvoSuite can be used for testing objectoriented programs usually, it can be applied only to structured mutation operators.

Accuracy

It is defined to be the ratio of the sum of the true positives and the true negatives, againstthe total number of classification parameters $(T_p + T_n + F_p + F_n)$.

$$\text{Accuracy} = \frac{Tp\ + Tn}{Tp + Tn + Fp + Fn}$$

Where,

Tp -True positive

Tn – Ture negative

Fp -False positive

Fn – False negative

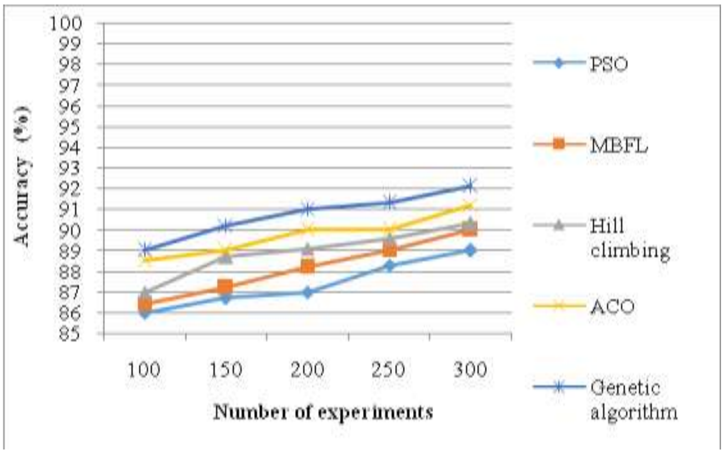

Figure 1 Accuracy comparison

Figure 1 shows that the comparison made of PSO, MBFL, hill climbing, ACO and genetic algorithm with regard to accuracy.The number of experiments is plotted along the X axis and along the y axis,accuracy is plotted. PSO, MBFL, hill climbing, ACO and genetic algorithm attains anaccuracy outcome of 89%, 90%, 90.3 %, 91.2 % and 92.1%. It is concludedthat the genetic algorithm has exhibited higher accuracy results.

Precision

Precision is defined to be the ratio of the true positives against both true positives and false positives results for intrusion and real features. It is expressed as below

$$\text{Precision} = \frac{T_p}{T_p + F_p}$$

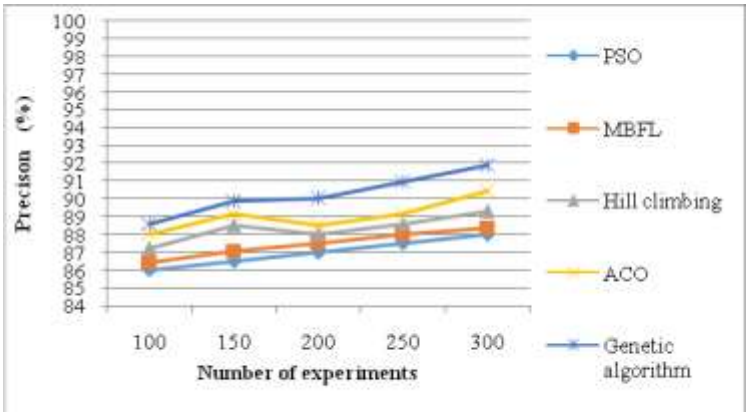

Figure 2 Precision comparison

Figure 2 illustrates the comparison result obtained from thealready availablePSO, MBFL, hill climbing, ACO and the newgenetic algorithm with regard to precision. The number of experimentsis plotted along the X axis and along the y axis precision is plotted. PSO, MBFL, hill climbing, ACO and proposed genetic algorithm attains the precision result of 88%, 88.4%, 89.3 %, 90.5 % and 91.9 % correspondingly. It has been found from the graph that the genetic algorithmperforms better than that of the rest of the models with regard to precision values.

Recall

It is the measure of the ratio of positives, which are correctly detected

$$\text{Recall} = \frac{T_p}{T_p + F_n}$$

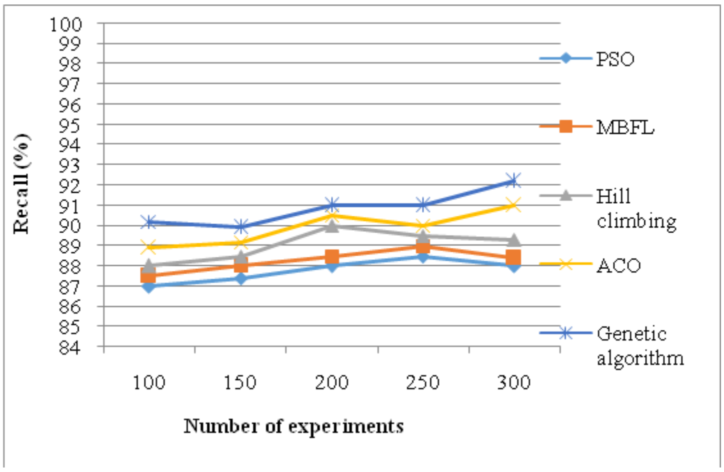

Figure 3 Recall comparison

Figure 3shows the comparison made of the availablePSO, MBFL, hill climbing, ACO and newgenetic algorithmwith regard to recall. The number of experimentsisplotted along the X axis and along the y axis recall is plotted.PSO, MBFL, hill climbing, ACO and the newgenetic algorithm techniqueattains recall result of 88%, 88.4%, 89.3 %, 91 % and 92.2 %correspondingly. It is concluded that the genetic algorithm has produceda higherrecallvalue.

IV. CONCLUSION

Optimization of mutation testing cost reductiontechnique andenhancement of the generation efficiency are very essentialforsoftware testing. It is employed to choose better test cases forthe generation of software. This research work provides the analysis ofdifferenttechniques fornovel software reliabilitytest techniques in object-oriented programming. The testing methodologies and risk factor analysis techniques by various authors has been studied. Those research techniques are explained along with their pros and cons in detail to discover the efficiency of each algorithm. The research has been compared with one another depending on their resulting metrics to uncover the better strategy to proceed the next research focus in future.

REFERENCES

[1.] Kan, Stephen H. Metrics and models in software quality engineering. Addison-Wesley Longman Publishing Co., Inc., 2002.

[2.] A.S Namin and J.H. Andrews, “Finding Sufficient Mutation Operators via Variable Reduction” in Proceedings of the 2nd workshop on Mutation analysis (MUTATION’06). Raleigh, North Carollna: IEEE Computer Society, November 2006, p. 5

[3.] P. Chevalley, "Applying Mutation Analysis for Object Oriented Programs Using a Reflective Approach," In Proceedings of the 8th Asia-Pacific Software Engineering Conference, Macau SAR, China, December 2001

[4.] Ma, Yu-Seung, Mary Jean Harrold, and Yong-Rae Kwon. "Evaluation of mutation testing for object-oriented programs."*Proceedings of the 28th international conference on Software engineering*. ACM, 2006.

[5.] Zhang, Wei, et al. "A Method of Software Reliability Test Based on Relative Reliability in Object-Oriented Programming." *Information Processing, 2009. APCIP 2009. Asia-Pacific Conference on*. Vol. 1. IEEE, 2009.

[6.] Huang, Ming, Chunlei Zhang, and Xu Liang. "Software test cases generation based on improved particle swarm optimization." *Information Technology and Electronic Commerce (ICITEC), 2014 2nd International Conference on*. IEEE, 2014.

[7.] Li, Liping, and HonghaoGao. "Test suite reduction for mutation testing based on formal concept analysis." *Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), 2015 16th IEEE/ACIS International Conference on*. IEEE, 2015.

[8.] Gong, Pei, Ruilian Zhao, and Zheng Li. "Faster mutation-based fault localization with a novel mutation execution strategy."*Software Testing, Verification and Validation Workshops (ICSTW), 2015 IEEE Eighth International Conference on*. IEEE, 2015.

[9.] Souza, Francisco Carlos M., et al. "Strong mutation-based test data generation using hill climbing." *Search-Based Software Testing (SBST), 2016 IEEE/ACM 9th International Workshop on*. IEEE, 2016.

[10.] Prajapati, Neha, and Naveen Kumar. "Data flow based quality testing approach using ACO for component based software development." *Computing, Communication and Automation (ICCCA), 2016 International Conference on*. IEEE, 2016.

[11.] Ma, Yu-Seung, and Sang-Woon Kim. "Mutation testing cost reduction by clustering overlapped mutants." *Journal of Systems and Software* 115 (2016): 18-30.

[12.] Bashir, Muhammad Bilal, and AamerNadeem. "Improved Genetic Algorithm to Reduce Mutation Testing Cost." *IEEE Access* 5 (2017): 3657-3674.

[13.] M. Wang, B. Li, Z. Wang, X. Xie, "An Optimization Strategy for Evolutionary Testing Based on Cataclysm", In the proceedings of 34th Annual Computer Software and Applications Conference Workshops, 2010.