Volume 5, No. 2, February 2014
Journal of Global Research in Computer Science
ISSN-2229-371X
REVIEW ARTICLE
Available Online at www.jgrcs.info

MUTATION TESTING: A REVIEW

Pawan Kumar Chaurasia
Assistant Professor
Department of Information Technology,
Babasaheb Bhimrao Ambedkar University,
(A Central University) Lucknow (U.P), India, 226025
pkc.gkp@gmail.com

***Abstract*—** Mutation testing is a fault based testing technique in which mutants are generated in the program and apply different test cases on the mutants. Some mutants are killed and some mutants are alive. On the base of killed and alive mutants, mutant score is calculated. Mutants are categorizing into weak, strong and firm mutants on the cost reduction methods. This paper is used to review the mutation testing and categorize the mutants and focus on cost reduction techniques.

***Keywords*-** Mutants, EX-WEAK, ST-WEAK, BB-WEAK, FOM, HOM, SSHOM, SUPER MUTANT.

INTRODUCTION

The purpose of software testing is a process of verifying and validating software applications or program so that it achieves the requirements and development design that expected by the project or user. The first mutation testing was proposed in 1970 and its tool was implemented by the Timothy Budd in 1980 in his research work [1]. According to Budd "Mutation testing is a fault based testing technique in which we seek the errors in the program and find the errors". These faults are seeded in the original program and compare with the mutated program. Mutants are introduced when the programs are started by the mutant operators. Each mutation produces a mutant program, produced by a mutation operator.

This paper is described in six phases of the mutation. First phase is to introduce about the mutation testing. Second phase is to calculate the mutation score after kill the mutants. Third phase is how the test cases are effective and calculate through a mutation score. Next phase is to define the weak mutation testing and then how mutants are reduced. Last phase is to calculate the cost of executed mutants.

MUTATION TESTING

Mutation testing is a method of software testing which is proposed by the Hamlet [3]. Mutation testing is a fault based testing technique [4,7,8]. It is a kind of testing in which the application is tested for the code that was modified after fixing a particular defect. In mostly cases test of a program that uncover simple errors are also effective in uncovering much more complex errors. The coupling effects can be used to save during the testing process. Mutations are based on operators are called mutant operators [5-6]. As in figure 1 mutants are generated by seeking some changes in the original program and generate the different test cases and execute the mutant program. If a test is different from the original program, it is said to kill the mutants. If the test case is not different between the mutant and the original program then the mutant is still live. A mutants remain alive because it is equivalent to the original program. If the mutant produce the same output, it can't be killed.

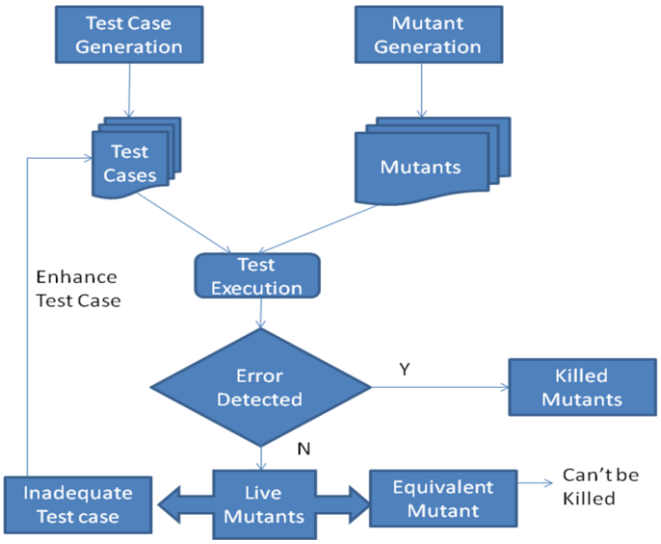

Figure 1. Mutation Testing Process

It is used to test the test case to kill all the mutants. Test cases are generated to kill all the mutants. A test set which can kill all non-equivalent mutants is said to be adequate and the mutant score which is measured by the total number of killed mutants over the non-equivalent mutants where a mutant is said to be equivalent. Mutant score takes real values between 0.0 and 1.0 which is the best score.

$$\text{Adequacy of test set} = \frac{\text{Number of killed mutants}}{\text{Number of non-equivalent mutants}}$$

EFFECTIVENESS OF TEST CASES

Test cases are effective by generating the relation between test cases effectiveness and mutation score with the equation presented [9]. It can be calculated by multiply the mutation score (M) and the ratio of average of number of test cases that kill mutants (K) divided by the total no of test cases (T).

$$E = (\text{Mutation score} * \text{average no of test cases}) / \text{total test cases}$$

To calculate, average number of test cases that killed mutants divided by number of dead mutants.

Average no of test cases= killed mutants / no of dead mutants

WEAK MUTATION TESTING

To reduce execution cost, Howden [10] proposed weak mutation testing, which only requires two (reachability, infection) conditions to kill a mutants. In weak mutation, the states of both programs are compared at a predetermined point after the execution of the mutated instruction. If the states are different at that point, the mutant is killed. Weak mutation testing is different from other mutation testing which focus on component in a program. Suppose that program A and B, where B is simple component of A and mutated version of B produces B'. So A' is mutated version of A containing B.

Howden does not describe the precise definition of program components in the original paper, but he further refined in the functional testing [11]. There are basically five types of components i.e Variable Reference, Variable Assignment, Arithmetic Expression, Relational Expression and Boolean Expression.

Woodward and Halewood's[12] define the component where the state of the original and mutated program are compared. Weak mutation testing is categorized into four types as follows.

EX-WAEK/1 (Expression-Weak/1) Mutation: It compares the state between an original program and mutant generated after the execution of the expression component.

ST-WAEK/1 (Statement-Weak/1) Mutation: It compares the state after the first execution of the mutant statement and the original program.

BB-WAEK/1 (Basic-Block-Weak/1 execution) Mutation: It requires check the states to be compared at the end of the first execution of the mutated statement. There are many mutants in the loop that could not be killed on the first execution that could be allowed to multiple execution till all the mutants are dead.

BB-WAEK/N (Basic-Block Weak/N execution) Mutation: It is the extended version of BB-Weak/1 that execute each block for the mutant to check the statement. When the mutants are checked it can't be killed in single execution of the statement and the statement is terminated.

From research work [13] and from the report of [14,10], weak mutation testing is to be generate mutants framework as in figure 2 which is applied on the mutants operator and weak mutation technique. The components and framework of real mutation testing are:

a. **Pre - Test Phase:** In this phase testers upload a program to test and validate it. Mutants are generated based on mutant operator and store the mutant moved to the database. Test cases and mutants are loaded from test case database, into test mutant controller.

b. ***Testing Phase:*** In this phase the timer start the execution time and the test controller deploy the mutant and test cases on the server. After testing the program server send the result and compare from the original program and the result are stored. If the mutants are killed and live that stored into the database.

c. ***Post Testing Phase:*** It is the last phase of the test framework. In this phase the results are stored into the database of the test case. On the base of database mutation score is calculated and check the effectiveness of the result after test the mutated program.

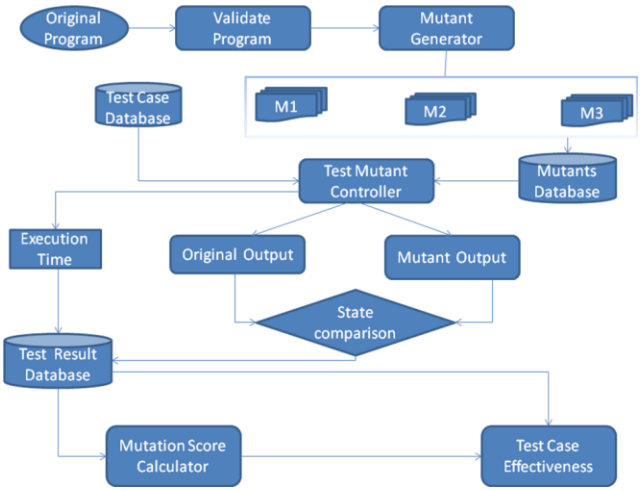

Figure 2. Mutation Framework

REDUCED MUTANTS

Mutation testing is one of the expensive testing technique. Major sources of computational cost in mutation testing is the inherent running cost in executing the large number of mutants against the test set. There are basically four types of techniques to reduced the mutants.

a. ***Mutant Sampling:*** It is a simple approach that randomly choose a small subset of mutants from the entire set of mutants. In mutation sampling all possible mutants are generated and select randomly for mutation analysis and the remaining are discarded.

b. ***Mutant Clustering:*** It is proposed by the Hussain Master Thesis [15]. Mutant clustering generate all first order mutants into different based on the killable test cases. Each mutant in the same cluster is killed by a similar set of test cases. Only a small number of mutants are selected from each cluster to be used in mutation testing and the remaining are discarded. Domain reduction techniques introduced by the Ji [16].

c. ***Selective Mutation:*** It seeks to find small set of mutation operators that generate a subset of all possible mutants without significant loss of test effectiveness. It was first proposed by Mathur[17]. Offut[18] extended the work by omitting four and six selective mutation operators. Based on Mothra mutation operators divide them into three categories: statement, operands and expressions. The most recent research work on selective mutant was Namin [19-21] by formulating the selective mutant problem into

statistical problem. They applied linear statistical approach and reduced 92% of all generated mutants.

d. Higher Order Mutant: Mutants can be divided into first order mutants (FOM) and higher order mutants(HOM). FOM are generated only once by applying mutant operator while in HOM are generated by applying mutation operator more than once. Jia and Harman introduced the concept of subsuming HOMs [22]. It is not easy to kill all the FOMs from which it is constructed. It is preferable to replace FOMs than the single HOMs to reduce the number of mutants. They also introduced the concept of strongly subsuming HOM (SSHOM) which is only killed by a subset of the intersection of test cases that kill each FOM from which it is constructed. It is partially proved by Polo et al [23].

COST REDUCTION TECHNIQUES

Reduced the number of mutants, the computational cost can also be reduced by reducing the mutant execution process. There are three techniques to reduce the execution cost that have been considered in literature.

a. Strong, Weak and Firm Mutation: Strong mutation is referred as traditional mutation testing which is proposed by DeMillo et al [4]. The mutants are killed only if the output is different from the original program. Optimize the execution of the strong mutation Howden[24] proposed a weak mutation instead of checking mutants after the execution of the entire program, the mutant only need to check immediately after the execution point of the mutant or mutated component. Advantage of weak mutant is that each mutant does not require a complete execution process; once the mutated component is executed we can check for survival mutants.

Firm mutation was first proposed by Woodward and Halewood[12] in 1988. In firm mutation, disadvantage of strong and weak mutation is removed. It lies between the after execution (weak mutation) and the final output (strong mutation) of the mutation. In 2001 Jackson and Woodward [25] proposed a parallel firm mutation approach for java programs.

b. Optimization techniques for runtime mutation: In first generation [26] testing tool, interpreter based technique is used to optimize the mutation, i.e the result of a mutant is interpreted from its source code directly. Mutant optimization is sufficient and efficient for small mutant programs. To reduce the cost of interpretation, compiler based technique was proposed [27], because execution of compiled binary code is much faster than interpretation. In compiled-based technique, the mutant program is compiled into an executable program, then each compiled mutant is executed by a number of test cases. High speed limitation due to high compilation cost for large programs [28]. DeMillo et al. proposed the new technique compiler-integrated to optimize the performance of traditional compiler [29]. The new approach of mutant schema generation reduced the overhead cost of traditional interpreter based [30-31]. It is not easy to compile all the mutants, instead of compiling each mutant separately, mutant schema generate a meta programs like “super mutant”. This meta program is need to compile once time to test each mutant. So the cost is calculated once time compilation and overall runtime cost. After compilation technique the new approach is introduced as a byte code translation technique which is proposed by the Ma et. al. [32].

CONCLUDING REMARKS

This paper introduce about the mutation testing and the process of mutation testing to find the live mutants and killed mutants. On the base of killed mutants mutant score is calculated. Test cases are effective by generating the relationship between test case and mutation score. On the base of test case mutant are defined as weak and strong mutants. Weak mutants are categorized into four parts as expression weak, strong weak, basic-block/1 and basic-block/N. Weak mutation testing is to generate mutant framework which is applied on the mutant operator and its technique. It is very expensive testing, for reducing the mutation testing. Various procedures are described in phase five. After reduced the mutants, computational cost are also reduced by optimize the runtime mutants of different techniques.

In future mutants are used for security policies to find the weak positions in security features. Efficiency of the program can also be increased by calculating the mutant operators. Mutants effectiveness can also be categorized into highly effective, effective and low effective.

REFERENCES

[1]. E.J Weyuker and T.J Ostrand, “Theories of Program Testing and the Application of Revealing Subdomains,” IEEE Transaction Software Engineering., vol. SE-6, 1980.

[2]. J. Goodenough and S. L. Gerhart, “ Towards a theory of Test Data Selection,” IEEE Transaction Software Engineering., vol. SE-3 1977.

[3]. R.G Hamlet, (1977), “Testing programs with the AID of a Compiler”, IEEE Transactions on Software engineering, 1977.

[4]. R. DeMillo, R. Lipton and F Sayward,(1978), “Hints on Test Data Selection: Help for the Practicing Programmer,” Computer, 11(4): 34-41: April, 1978.

[5]. Antonia Estero-Botaro Palomo-Lozano and Inmaculada Medina Bulo, “Quantitative Evaluation of Mutation Operators for WS-BPEL Compositions” Department of Computer Languages and Systems, University of C? adiz, Spain.

[6]. M.Woodward,“ Errors in Algebaric Specification and an Experimental Mutaion Testing Tool” Software Engineering Journal, pages 211-224, July 1993.

[7]. Y. Jia and M. Harman, “An Analysis and Survey of the Development of Mutation Testing”, CREST Center, King’s College, London, Tech. Rep. TR-09-06, 2009.

[8]. A.J. Offut., "Investigations of the Software Testing Coupling Effect", ACM Transactions on Software engineering Methodology 1(1):3-18 January 1992.

[9]. A . Derzinska, "Quality Assessment of Mutation Operators Dedicated for C# Programs" in QSIC 2006: sixth International Conference on Quality Software, Beijing, China: IEEE, Computer society , 2006, pp 227-234.

[10]. Howden W. E. (1982), "Weak Mutation Testing and Completenes of Test Sets", IEEE transaction on Software Engineering, 8(4): page 371-379.

[11]. W. E Howden, "Functional Programming Testing and Analysis", McGraw-hill Book company New York NY 1987.

[12]. M.R Woodward and K. Halewood, (1988), "From Weak to Strong, Dead or Alive? An analysis of some mutation testing issues", Workshop on Software Testing, Verification and Analysis, pages 152-158, Banff Alberta, July 1988. IEEE Computer Society Press.

[13]. Natthapol Thaisakonpun and Taratip Suwannasart, "Mutation Testing for Expression Modification Operator of BPEL" Software Engineering Laboratory, Centre of Excellence in Software Engineering, Faculty of Engineering, chulaongkorn University, Bangkok, Thailand.

[14]. A. Jefferson Offut and Stephen D. Lee, "An Empirical Evaluation of Weak Mutation", Department of Information of Informal and Software Systems Engineering, George Mason University Fairfax, VA 22030, Stephen D. Lee, IBM corporation A00/062, P.O Box 12195, Research Triangle Park, NC 27709, February 24, 1996.

[15]. S. Hussain, "Mutation Clustering" Masters Thesis, king's college London, strand London, 2008.

[16]. C. JI. Z Chen, B. Xu and Z. Zhao, "A Novel method of Mutation Clustering Based on Domain Analysis" in Proceedings of the 21st International Conferenceon Software Engineering and Knowledge Engineering (SEKE'09). Boston, Massachusetts: Knowledge Systems Institute Graduate School, 1-3 July 2009.

[17]. A.P. Mathur, "Performance Effectiveness and Reliability Issues in Software Testing," in Proceedings of the 5th International Computer Software and Applications Conference (COMPASC'79), Tokyo, Japan, 11-13 September 1991, pp 604-605.

[18]. A. J Offut, G. Rothernel and C.Zapf, "An Experimental Evaluation of Selective Mutants," in Proceedings of the 15th International Conference on Software Engineering (ICSE'93). Baltimore, maryland IEEE Computer Society Press May 1993, pp100-107.

[19]. 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.

[20]. A.S Namin and J.H. Andrews, "On Sufficiency of Mutants" in Proceedings of the 29th International Conference on Software Engineering (ICSE ComPANION'07) Minneapolis, Minnesota, 20-26 May, 2007, pp. 73-74.

[21]. A.S Namin and J.H. Andrews and D. J. Murdoch, "Sufficient Mutation Operators for Masuring Test Effectiveness" in Proceedings of the 30th International Conference on Software Engineering (ICSE'08), Leipzig, Germany, 10-18 May 2008, pp. 351-360.

[22]. Y.Jia and M. Harman, "Constructing Subtle Faults Using Higher Order Mutation Testing," in Proceedings of the 8th International Working Conference on Source code Analysis and Manipulation (SCAM'08), Beijing, China, 28-29 Sepetember 2008 pp 249-258.

[23]. M. Polo, M. Plattini and I Garcala-Rodriguez "Decreasing the Cost of Mutation Testing with Second Order Mutants" Software Testing Verification and Reliability, vol. 19, no. 2, pp. 111-131, June 2008.

[24]. Y. Jia and M. Harman, "Higher Order Mutation Testing" Journal of Information and Software Technology, King's College London, CREST Centre Strand, London, March 11, 2009, pp 1-41.

[25]. D. Jackson and M.R. Woodward, "Parallel Firm Mutation of Java Programs" in Proceedings of the 1st Workshop on mutation analysis (MUTATION'00), published in book form, as Mutation Testing for the New Century, san Jose, California, 6-7 October 2001, pp 55-61.

[26]. A.J. Offutt and K. N. King, "A Fortran 77 Interpreter for Mutation Analysis" ACM SIGPLAN Notices, vol. 22, no. 7, pp. 177-188, July 1987.

[27]. M.E. Delamaro, "Proteum- A Mutation Analysis Based Testing Environment" Masters Thesis, University ofbSao Paulo, Sao Paulo, Brazil, 1993.

[28]. B. Chol and A.P. Mathur, "High Performance Mutation Testing" Journals of Systems and Software, vol. 20, no. 2, pp. 135-152, February 1993.

[29]. R.A. DeMillo, E.W. Krauser and A.P. Mathur, "Compiler-Integrated Program Mutation", in Proceedings of the 5th Annual Computer Software and applications Conference (COMPASC'91), Tokyo, Japan: IEEE Computer Society Press, September 1991, pp. 351-356.

[30]. R.H. Untch, " Mutation Based Software Testing Using Program Scemata" in Proceedings of the 30th Annual Southeast Regional Conference (ACMSE'92), Raleigh, North Carolina, 1992, pp. 285-291.

[31]. R.H. Untch, "Schema-Based Mutation A New Test Data adequacy Assessment Method" Ph.d Thesis, Clemson University, Clemson, south Carolina, December 1995.

[32]. Y.S. Ma., A.J. Offutt and Y.R. kwon, "MuJava: An automated Class Mutation System" Software Testing, Verification and Reliability, vol. 15, no. 2, pp. 97-133, June 2005.

