Survey on Different Approaches for Mutation Testing

Jyoti J. Danawale
PG Student
D.Y.Patil College of Engineering,
Ambi, Talegaon

Sandeep Kadam
Head of Department
D.Y.Patil College of Engineering,
Ambi, Talegaon

ABSTRACT

One of the effective techniques for testing is mutation testing. Mutant can be created by changing the syntax of a program. To distinguish the mutant from the original program, an effective test suite is required. The Mutation testing is a testing method aimed at improving the adequacy of test suites and estimating the number of faults present in systems under test. The mutations can be applied to the source code and the semantics of the language. The mutations of the semantics of the language signify possible misunderstandings of the description language and thus capture a different class of faults. As the possiblemisunderstandings are highly context reliant, this context should be used to determine which semantic mutants should be formed. The approach is illustrated through examples and code in php. In addition, a semantic mutation testing tool for Php is proposed.

Keywords

Mutation Testing, Semantic Mutation Testing, Scripting Language PHP

1. INTRODUCTION

Software testing is the process of execution of a program or application with the intent of finding the bugs in the software. Software testing can also be specified as the process ofvalidation and verification that a software program or application or product is working as expected or not. Testing is the process of evaluating a system or its component with the intent of finding whether it satisfies the specified requirement or not. Testing is the execution of a system to recognize any gaps, errors, or missing requirements in opposing to the actual requirements [2]. Testing performed by a developer on completion of the code is also considered as testing. There are two classes of troubles in Computer Software: faults or failures. Fault is a condition that causes software to fail to perform its required function. The error is the difference between Actual and the Expected output. Failure is the in ability of a system or a part of a system to perform required function as per its specification.Testing a software helps in comparing the application or product against user and business requirements. It is most important to have good test coverage to test the software application completely and to ensure that it performs well and according to the SRS.If the test case coverage of the code is high, the test cases have to be very strong with maximum cases of detecting the faults. This objective can be calculated by taking into consideration the count of defects reported per test case. More the number of defects, it means the test cases were made very strong.

2. MUTATION TESTING

Mutation testing is a type of testing aimed at locating and exposing the weakness in test suites. The main motivation for Mutation testing is to make strong test cases in contrast of finding faults in the source code. Mutation testing aims on strength of test suites which is used for checking the source code it falls under the category of white box testing sincethe source code is available to us for testing [9]. Mutation testing is also referred as fault based testing. Any small change that differentiates the program from the original program is a mutant. There are various types of mutants: stillborn, trivial, equivalent.The prerequisite is the source code and a test case for testing that source code. To create amutant, only thing that is required is to vary the original program by inserting a smallfault in it. The mutants are checked by running the original test data. Differences refer themutant are killed. In case the mutant remains live, the possibility arises either ifthe mutant and the native program are equivalent or the mutant could not be killed as thetest set was inadequate. Traditional mutation testing consists of operators for a mutationthat represent syntactically small errors like replacing + by – in an arithmetic expression.There are several drawbacks of traditional mutation testing. Some of the drawbacks arelisted: for a small program, the number of mutants produced is large, whichincreases the chances of equivalent mutants. To deal with equivalent mutants, the extraamount of manual work is required. This extra effort increases the cost of testing.Mutation operators do not consider the misunderstandings related to semantic changes;the only concern is syntactic level.Mutation testing is based on two premises. The first one amongst them is the competent programmer hypothesis. Most of the faults which occur in the application code are due to syntactic errors. This is the agenda on which competent programmer hypothesis is based upon. Coupling effect is the second hypothesis. The coupling effect says that the test data or the test case which is able to detect simple type of bugs are good enough for detection of complex defects. When more than a single change is made in the code we get higher order mutants. Mutation testing is accomplished by first making theOperators. These operators are known as mutation operators. Then the test case is given input against the original and the varied code. After that the results are compared for the both of them. If the output comes out to be different from the original one, then the mutant is said to be killed.

3. SEMANTIC MUTATION TESTING

To deal with several specific types of mistakes, Semantic Mutation Testing was proposed [18]. The semantics can even be changed by a small change in the syntax. For introducing the semantic mistakes, different ways are available. For SMT-P, change in the syntax of description has been chosen in orderto simulate semantic mutation. Three types of MT can be studied under SMT-P i.e. weak MT, Firm MT and strong MT.

In case of strong mutation testing, the program, and the mutant can be separatelyidentified, if they produce different outputs for a same test case. On the other hand, inweak mutation testing, if the program and the mutant reflect a value which is not same for a variable immediately after the particular point at which the program was mutated are said to be distinguished.With the help of Firm MT we can in general allow the quality tester or the debugger to select the position at which the value of a variable can be changed. In SMT the semantics of that particular language is denoted with the symbol 'L', and in totality the behavior is described by the use of pair i.e. (N,L). If there is any change in the traditional form of MT then the description of mutant will be changed to (N', L). But in case we alter the semantics of the given language the description would be given as (N,L').

4. RELATED WORK

Mutation testing Research focuses on three kinds of activities such as defining mutation operators, conducting tests and implementing tools. The first activity involves designing new mutation operators for differentlanguages. The second research activity is testing with mutations. Empirical studies have supported the efficiency of mutation testing. Mutation testing more powerful than statement and branch coverage and more efficient in finding faults than data flow. Offutt et al. and Wong and Mathur[4] evaluated the idea of selective mutation that identifies the critical mutation operators and provide almost the same testing coverage as non-selective mutation. Using this approach considerably decreases the number of mutants generated and hence reduces computational cost. The third activity in mutation testing research is implementing mutation tools. Mothra and Proteum were developed for Fortran and C, respectively. Jester, Jumble and MuJava, as well as the tool presented in this paper, arecommitted to the Scripting language. An important feature of mutation testing tools is the mutation operators supported by a tool. Here the work has been done with two types of mutation operators: (1) traditional mutation operators tailored from procedural languages and (2) Semantic mutation operators designed.

5. TECHNIQUES TO OVERCOME MUTATION TESTING PROBLEMS

5.1 Techniques to reduce number of mutants

Four mutant reduction techniques have been proposed that willfit Mutation Testing into a practical testing technique.

A) Mutant Sampling*:* Mutant Sampling is a simple technique that randomly chooses a small subset of mutants from the entire set. In this approach, all possible mutants are generated[17]. Some of these mutants are then selected randomly for mutation analysis and the remaining mutants are thrown away.

B) Mutant Clustering: Mutant Clustering takes a subset of mutants using clustering algorithms.Mutation clusteringtechnique generates first order mutants. A clustering algorithm is then appliedon first order mutants to classify them into different clusters and there is guarantee that each mutant of the same cluster is killed by the related set of test cases. Only a small number of mutants are chosen from each cluster to be used in Mutation Testing, the left over mutants are excluded.

C) Selective Mutation: Number of mutants can also be reduced by reducing applied mutation operators. Selective Mutation finds a small set of mutation operators that generates a subset of all possible mutants without major loss of test efficiency.

D)Higher Order Mutation: Higher Order Mutation is a somewhat new form of Mutation Testing [11]. The objective of this technique is to find out respected higher order mutants. First Order Mutants are produced by applying a mutation operator only once. Higher Order Mutants are created by applying mutation operators more than once.

5.2 Cost Reduction Techniques

The computational cost can be reduced by improving the mutant execution process. Two cost reduction techniques are proposed.

A) Strong, Weak and Firm Mutation: Strong Mutation Testing is also known as traditional mutation testing and it is proposed by DeMilo et al. In Strong Mutation, for a given program p, a mutant m of program p is supposed to be destroyed or killed only if mutant m gives a dissimilar output from the original program p. Weak Mutation is proposed to optimize the execution of strong mutation. In weak mutation it is assumed that a program p is constructed using a set of components C={c1,c2,c3,….,cn}. Suppose mutant m is made by changing component cm then mutant m is said to be killed if execution of component cm is dissimilar from mutant m. In weak mutation the mutants arechecked immediately after the execution point of mutant instead of checking mutants after execution of whole program.Previous work is carried out in Firm Mutation by Woodward and Halewood. Woodward and Halewoodhave suggested techniques to minimize the drawbacks of both weak and strong mutation. For this they have used the concept of continuum of intermediate possibilities.

B)Run-Time Optimization Techniques: Interpreter-based technique is used by the first generation of Mutation Testing tools to optimize the mutation. The result of a mutant is interpreted from its source code directly. Compiler-based technique was proposed to reduce the cost of interpretation as the execution of compiled binary code is faster than interpretation. In compiler-based technique, the mutant program is compiled first and then numbers of test cases are applied to the mutant program.

6. PROPOSED SYSTEM

A Mutant Generation Tool is proposed for scripting language PHP. The tool consists of three modules: Mutant Generator, Mutant Executor and Run Test cases, Calculate efficiency of test suite.

A) Mutant Generator: The input to the mutant generator is the original program and the mutation operators. The mutant generator will then generate the mutants by using guided mutation testing algorithm.

B) Mutant Executor: The output from Mutant Generator will be input to the Mutant Executor. PhpUnit is the unit testing framework in PHP language. The test cases are generated and applied to the Mutant Executor. Mutant Executor runs test cases.

C)Calculate Efficiency of test suite: Mutant Executor runs the test cases and based on that result, the efficiency of the test suit is measured. Atest case is said to be killed if it gives different result when applied to the original program and the mutant program.If a test case does not pass, it means that the test case strength is high and the mutant is said to be killed.

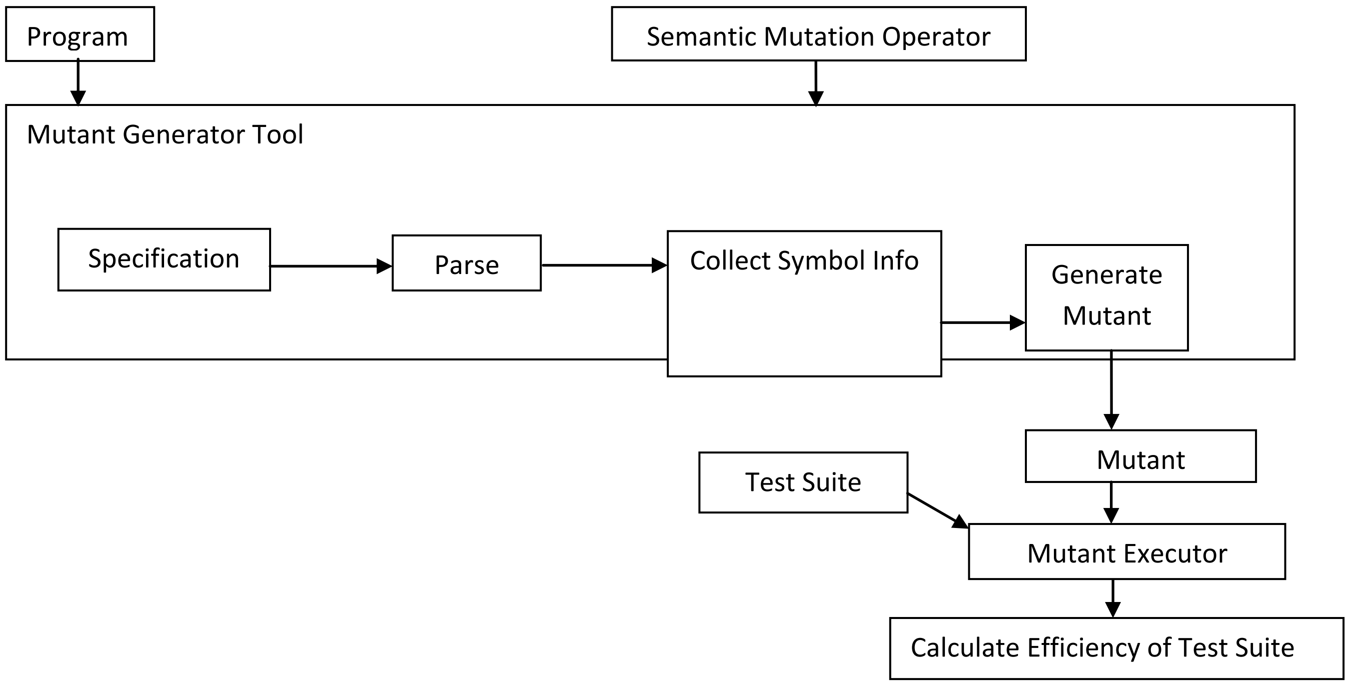

Fig1. Proposed System Architecture

7. CONCLUSION

One of the effective techniques for testing is mutation testing. Mutant can be created bychanging the syntax of a program. To distinguish the mutant from the original program,an effective test suite is required.A new tool for scripting language Php is proposed based on semantic mutation. In semanticmutation, the language is modified to produce the mutant. There can bemisunderstandings in regard to the semantics of the description language. When thesyntax of a description is mutated, it is traditional mutation testing. On the other hand, when we deal with language, it is semantic mutation testing. A test case that producesdifferent results when run on the actual program and its mutant is said to be failed. Whena test case fails, the mutant is said to be killed. In future mutation testing can be applied to real time applications.

8. REFERENCES

[1] Umar M., “An evaluation of Mutation Operators for Equivalent Mutants”, M.S. thesis, Computer Science, King’s College, london, 2006.

[2] Beizer B., “Software testing techniques”, Dreamtech Press, 2003.

[3] Pressman, Roger S., “Software Engineering: a practitioner’s approach”, Pressman and Associates, 2005.

[4] A. J. Offutt, A. lee, G. Rothermel, R. Untch, and C. Zapf. “An experimental determination of sufficient mutation operators”. ACM Transactions on SoftwareEngineering Methodology, 5(2):pp. 99–ll8, April l996.

[5] J. Tuya, M. Suarez-Cabal, and C. de la Riva, “Mutating database queries,”Information and Software Technology, vol. 49, no. 4, 2007, pp. 398 – 4l7.

[6] D. Baldwin and F. G. Sayward, “Heuristics for determining equivalence of program mutations," tech report 276, Yale University, New Haven, Connecticut, l979.

[7] sDu Bousquet and M. Delaunay, “Towards mutation analysis for lustreprograms". Electronic Notes in Theoretical Computer Science, vol. 203, no. 4, pp. 35-48, 2008

[8] B. J. M. Grun, D. Schuler, and A. Zeller, “The impact of equivalent mutants," inProceedings of the IEEE International Conference on Software Testing, Verification, and Validation Workshops, (Denver, Colorado, USA), pp. l92-l99, IEEEComputer Society, 2009.

[9] M. Harman, R. Hierons, and S. Danicic, “The relationship between program dependence and mutation analysis," in Mutation testing for the new century (W. E. Wong, ed.), Norwell, MA, USA: Kluwer Academic Publishers, pp.5-l3, 200l.

[10] M. Woodward and K. Halewood, “From weak to strong, dead or alive? An analysis of some mutation testing issues", in Software Testing, Verification, and Analysis, l988., Proceedings of the Second Workshop on, pp. l52-l58, Jul l988.

[11] Y. Jia and M. Harman, “Higher order mutation testing," Information and Software Technology, vol. 5l, pp. l379-l393, Oct 2009.

[12] M. Kintis, M. Papadakis, and N. Malevris, “Evaluating mutation testing alternatives: A collateral experiment," in Proc. l7th Asia Pacific Software Engineering Conf. (APSEC), pp.300-309, 20l0.

[13] J. Ofutt and Y.-R. Kwon, “The class-level mutants of MuJava," in Proceedings of the 2006 international workshop on Automation of software test - AST '06,AST '06, (New York, New York, USA), pp. 78-84, ACM Press, 2006.

[14] M. Papadakis and N. Malevris, “An empirical evaluation of the first and second order mutation testing strategies," in Proceedings of the 20l0 Third InternationalConference on Software Testing, Verification, and Validation

Workshops, ICSTW 'l0, , IEEE Computer Society pp. 90-99, 20l0.

[15] D. Schuler and A. Zeller, "Uncovering equivalent mutants," in Proceedings of the 3rd International Conference on Software Testing Verification and Validation (ICST'l0), (Paris, France), pp. 45-54, Apr 20l0.

[16] Abdul Azim Abdul Ghani and Reza Meimandi," Aspect-Oriented ProgramTesting: An Annotated Bibliography", journal of software, vol. 8, no. 6, june20l3.

[17] YueJia and Mark Harman "An Analysis and Survey of the Development of Mutation Testing", IEEE Transactions On Software Engineering, vol. 7, no. 2, pp.77-84,2006.

[18] J.A. Clark, H.Dan and R.M. Hierons ," Semantic mutation testing", Science of Computer Programming pp:345-363, 20l3.

[19] Haitao Dan and Robert M. Hierons "SMT-C: A Semantic Mutation Testing Tool for C" IEEE Fifth International Conference on Software Testing, Verification and Validation, 20l2.

[20] M.Patrick, Manuel Oriol and John A. Clark "MESSI: Mutant Evaluation by Static Semantic Interpretation IEEE Fifth International Conference on Software Testing, Verification and Validation" 20l2.

[21] S.Singh and S.Jain "A study on equivalent mutants detecting technique", VSRD International Journal of Computer Science & Information Technology, Vol. 3 No.5 May 20l3.