Indian Journal of Science and Technology, Vol 9(10), DOI: 10.17485/ijst/2016/v9i10/88983, March 2016
ISSN (Print) : 0974-6846
ISSN (Online) : 0974-5645

Novel Approach for Whole Test Suite Generation using Metamorphic Relations

Ramya Bandaru and J. Albert Mayan

Department of CSE, Faculty of Computing, Sathyabama University, Chennai-600119, Tamil Nadu, India; ramya.05it@gmail.com, albertmayan@gmail.com

Abstract

Background: The software or an individual program will not get crash by the minute bugs in the code and always manual method for testing the code was not feasible, most of the cases the tester will adds the test oracles to the test cases using the manual method but it is not optimal solution for the large programs and software's and this method can targets only covering the one goal at a time. There is a problem with this coverage goals due to these goals are not independent. **Methodology:** To get out from these problems we propose a unique approach, in this approach we are generating the test cases automatically and developed an integrated method for program correctness, testing and debugging. **Findings:** We developed an unique approach in order to solve the oracle problem by using the metamorphic testing this approach also address the automatic debugging this testing uses the synergy algorithm, it does not attempt to traverse the execution tree, instead it attempts to cover all abstract states. **Applications/Improvement:** With metamorphic relations the system is ideal for medium and large scale applications and this approach also uses the fuzzy logic to provide the result whether the test case pass or fail.

Keywords: Automatic Debugging, Fuzzy Logic, Metamorphic Relations, Oracle Problem

1. Introduction

Testing is the crucial component while developing the software and it is widely used in order to make software process successfully. Input expected output and actual output are the main components for the testing the main use of input is to check the correctness of the program whether the resultant output was matching with the actual output and it also checks the definition for the expected result. Many techniques has been proposed over the years in order to generate the test cases automatically and generating test suites automatically is also proposed in recent years the test suite was the collection of test cases the main aim for the test suite is covering the maximum code still the problem in the test suite was the goal for one test case may be dependent on the other test case so the goal for the small test suite was not completely achievable so it leads to the oracle problem. Because it is easy for the tester to specify expected outcome in terms of oracle if the test suite was small otherwise it is very tough for the tester generate the oracles so, our aim is also to make oracles generation easy and to make the test suite should be applicable for the complex programs.

With regression testing cost was more and time consumption was also very high. By using test suite prioritization, we can prioritize the test suite for high fault detection and to reduce the time consumption[1]. Frame works are available that checks whether test data was violating the given metamorphic relations by using the constraint logic programming technique. There are so many techniques that are based on the assumption that complete oracle were available[2]. Context classification in feature clustering used for classifying xml document[3]. In order to generate the test oracles automatically it is

*Author for correspondence

very important to generate expected output automatically and also verifying the result from should also be done automatically is required for that we are using different software models[4]. By using software agent and fuzzy logic it determines the prioritization order by using prioritization system there by effectiveness was increased and also fault detection also gets increased[5]. The approach that followed commonly in this with the goal is to generate the test oracles automatically for the test cases by using some techniques such as metamorphic relations, modeling, contract driven development and specification but sometimes it will not give the adequate solution for the problem such time final choice was the human needs to specify the test oracles for the test cases. But it is difficult for human to specify the test oracles for test suites that covers the large amount of code it will be more time consuming process for the human tester. Because predicting the size of the test suite was very difficult[6].

Test suite size can be reduced by removing some of the test cases, it will happen when code modification was done over time but it causes the fault detection effectiveness[7]. Oracle was mainly used to determine pass or failure of the test cases while testing[8]. There are machine learning techniques are also available in order to generate the test oracles automatically[9]. Different kinds of genetic algorithms also available to automate the testing process and it also makes this automation fast[10]. It is possible even by the in experienced testers with less period of training to identify the sufficient metamorphic relations with appropriateness[11].

In previous studies, tester manually identifies the metamorphic relation through ad hoc way recent days new methods were also proposed for constructing the metamorphic relation based on other metamorphic relation that identified already so thereby it reduces the cost[12]. This algorithm also helpful for prioritizing the suites there by time consuming will be very less and increases the effectiveness[13]. There is an empirical study assesses the metamorphic relations quickly and also checks its usefulness, but it is very difficult evaluate the relation based on their usefulness mutation analysis is used to assess the metamorphic relation[14]. Size of the test cases in test suite can be modified by alleviating the test cases it was possible when there is modification done on the code[15].

There are different types of rules that are available to assess the better metamorphic relations[16]. Test case prioritization was also beneficial for random coverage as well as for structural coverage[17]. Fitness function uses genetic algorithm to improve the effectiveness of the fault detection[18]. The objective for the test suite not only the code coverage because whenever there is an modification or changes occur in the program mostly it violates code coverage but also test suite must be flexible for the evolution situation, like extension, removal and refactoring of test case and testing had been done for all scenarios[19]

Code coverage criteria was also very important aspect in testing, adaptive random testing and the random testing are two different types of testing both selects the test case in random with main goal of higher code coverage[20]. Generating test oracle automatically will reduce the cost nearly thirty five to fifty percent. Test oracle plays an important role in the soft ware testing because it checks about the behavior of software. Manual process for test oracle will be time consuming and more costly[21]. Genetic algorithm was also used in order to reduce the size of test suite it will be helpful to reduce the cost for some types of testing methods. The reduced test suite by applying genetic algorithm maintains the coverage criteria this algorithm uses the mathematical model to reduce suite size. In order to take decision regarding the reduction of test suite genetic algorithm uses text execution cost and block based coverage criteria[22].

There are several techniques that are available to detect better metamorphic relations, there by it increases the high fault detection[23]. Test Oracle can be automated by different types of method that are available widely[24]. Whenever test cases are added to existing test suite it increases size of suite it leads to increase of cost for testing such as regression testing in order to overcome this problem some of the algorithms for test suite reduction was implemented with the goal for reducing the cost as well as maximum code coverage criteria[25].

One of the main problem in software testing is the oracle problem i.e., if there is no mechanism called oracle to check the output for the given input it is called an oracle problem this problem can be overcome by using the metamorphic testing, metamorphic relations will get derive from the in born characteristic of software so, it is helpful to check the correctness of output without using the oracle. But adaptive random testing uses less test case when compare with the random testing not only that it also have higher code coverage thereby failure detection was also more but in case if the randomly generated test

cases missing any crucial part of program to test then it leads to a very big problem[26].

2. Existing System

The existing system was more concentrated on generating the small test suites with high code coverage but still the oracle problem was not alleviated and it is fail to generate the automatic assertions and in this system the test cases were less readable. And the system was developed in a tool it is fail to generate sufficient number of test cases because its main aim is to generate small size test suite which is collection of test cases.

3. Proposed System

In this paper we propose a metamorphic testing which derives metamorphic relations from the inborn properties of the program which lessen the oracle problem in the testing. Metamorphic testing identifies the some important properties of software or program and it represents these properties in the form of relations called Metamorphic relations each metamorphic relation in the metamorphic testing includes multiple set of inputs and expected outputs after testing it cross checks the results to confirm whether the metamorphic relations are satisfied or not.

Figure 1 depicts about how metamorphic testing applied by using the metamorphic relations. By using input space I set of metamorphic relations was derived and it checks whether the expected output was matching with the actual output there by it reduces the time to find if any errors or bugs in the program so, by using metamorphic relation we can alleviate the time constraint also

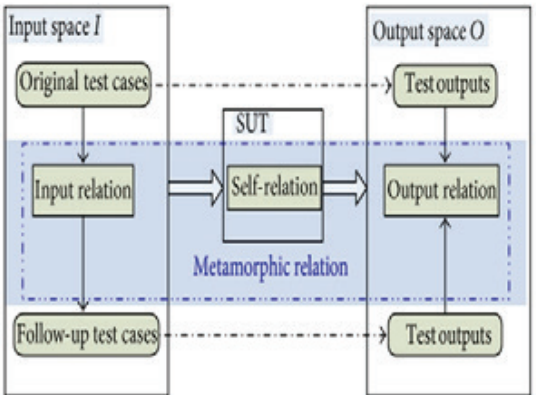

Figure 1. Metamorphic Testing.

Algorithm Used:

```
int Med (int u, int v, int w) {
   int med;
1  med = w;
2  if (v < w)
3     if (u < v)
4        med = v;
      else {
5        if (u < w)
6           med = u; }
   else
7     if (u > v)
8        med = v;
      else {
9        if (u > w)
10          med = u; }
11 return med; }
```

Figure 2. Pseudo code for Depth Test case Generation.

Metamorphic testing watches that regardless of the possibility that the executions don't bring about disappointments, despite everything they bear helpful data. Subsequent experiments ought to be built from the first arrangement of experiments with reference to chose vital properties of the predetermined capacity. Such important properties of the capacity are called Changeable relations. Typical examination and way limitation improvements alongside Metamorphic testing serves to incorporate the three distinct phases of SDLC subsequently empowering a robotized advancement environment that is both hearty and quick. Changeable testing watches that regardless of the possibility that the executions don't bring about disappointments, despite everything they bear helpful data. Subsequent experiments ought to be built from the first arrangement of experiments with reference to chose vital properties of the predetermined capacity. Such important properties of the capacity are called changeable relations. Typical examination and way limitation improvements alongside Metamorphic testing serves to incorporate the three distinct phases of SDLC subsequently empowering a robotized advancement environment that is both hearty and quick.

4. Fuzzy Oriented Metamorphic Relations

Specifically, we characterize the MRs that we expect characterization calculations to show, and characterize them all the more formally as takes after.

MR-0: Consistence with relative change. The outcome ought to be the same on the off chance that we apply the same discretionary relative change capacity, f(x) = kx + b, (k 6= 0) to each quality x to any subset of components in the preparation information set S and the experiment ts.

MR-1.1: Permutation of class marks. Expect that we have a class-mark change capacity Perm () to perform coordinated mapping between a class name in the arrangement of names L to another name in L. In the event that the source case result is li, applying the change capacity to the arrangement of relating class marks C for the subsequent case, the consequence of the subsequent case ought to be Perm (li).

MR-1.2: Permutation of the property. On the off chance that we permute the m traits of the considerable number of tests and the test information, the outcome ought to stay unaltered.

MR-2.1: Addition of uninformative characteristics. A uninformative property is one that is just as connected with every class name. For the source data, assume we get the outcome ct = li for the experiment ts. In the subsequent info, we add a uninformative credit to S and separately another characteristic in st. The decision of the real esteem to be included here is not critical as this characteristic is similarly connected with the class marks. The yield of the subsequent experiment ought to still be li. **MR-2.2:** Addition of educational characteristics. For the source information, assume we get the outcome ct = li for the experiment ts. In the subsequent data, we add an instructive credit to S and it's such that this property is firmly connected with class li and similarly connected with every single different class. The yield of the subsequent experiment ought to still be li.

MR-3.1: Consistence with re-expectation. For the source information, assume we get the outcome ct = li for the experiment ts. In the subsequent information, we can affix ts and ct to the end of S and C separately. We call the new preparing dataset S' and C'. We take S', C' and ts as the data of the subsequent case, and the yield ought to still be li.

MR-3.2: Additional preparing specimen. For the source information, assume we get the outcome ct = li for the experiment ts. In the subsequent information, we copy all specimens in S and L which have mark li. The yield of the subsequent experiment ought to still be li.

MR-4.1: Addition of classes by copying examples. For the source data, assume we get the outcome ct = li for the experiment ts. In the subsequent info, we copy all specimens in S and C that don't have mark li and link a self-assertive image "*" to the class names of the copied examples. That is, if the first preparing set S is connected with class names <A, B, C> and li is A, the arrangement of classes in S in the subsequent info could be <A, B, C, B*, C*>. The yield of the subsequent experiment ought to still be li. Another subsidiary of this transformative connection is that copying all examples from any number of classes which don't have mark li won't change the yield's consequence of the subsequent experiment.

MR-4.2: Addition of classes by re-naming examples. For the source info, assume we get the outcome ct = li for the experiment ts. In the subsequent data, we reliable a specimens' percentage in S and C which have mark other than li and connect a discretionary image "*" to their class names. That is, if the first preparing set S is connected with class marks <A, B, B, B, C, C, C> and c0 is A, the arrangement of classes in S in the subsequent information may get to be <A, B, B, B*, C, C*, C*>. The yield of the subsequent experiment ought to still be li.

MR-5.1: Removal of classes. For the source information, assume we get the outcome ct = li for the experiment ts. In the subsequent data, we uproot one whole class of tests in S of which the name is not li. That is, if the first preparing set S is connected with class marks <A, A, B, B, C, C> and li is A, the arrangement of classes in S in the subsequent info may get to be <A, A, B, B>. The yield of the subsequent experiment ought to still be li.

MR-5.2: Removal of tests. For the source info, assume we get the outcome ct = li for the experiment ts. In the subsequent info, we uproot some piece of a specimens' percentage in S and C of which the name is not li. That is, if the first preparing set S is connected with class names <A, A, B, B, C, C> and li is A, the arrangement of classes in S in the subsequent data may get to be <A, A, B, C>. The yield of the subsequent experiment ought to still be li.

5. Performance Evaluation

Fuzzy MR utilizes typical inputs and, consequently, involves nontrivial exertion in its execution, though Trans formative testing uses solid inputs and is, along these lines, much simpler to actualize. Semi-demonstrating, be that as it may, has novel points of interest. Firstly, not all inputs that practice a disappointment way may essentially bring about a disappointment. Regardless of whether a solid experiment meets the trigger condition is simply by shot. Then again, semi-demonstrating ensures the disappointment's identification. Also, if the project under test is right, transformative testing with solid inputs can just show that the system fulfills the changeable connection for the limited arrangement of tried inputs. Semi-demonstrating, then again, can exhibit the fulfillment of the transformative connection for a much bigger arrangement of inputs, which may be a vast set. This gives a higher certainty. Moreover, semi-demonstrating can be further consolidated with routine testing to extrapolate the program's rightness to related untested inputs. Thirdly, when a disappointment is distinguished, semi-demonstrating will give symptomatic data to investigating as far as imperative expressions, however transformative testing not has this capacity.

Based on the highlighted debugging information in Figure 2, our debugger further highlights a total of 15 lines of source code for inspection. These lines of code are not shown in this paper. Version 14 has a total of 225 lines of source code excluding blanks and comments. The number of highlighted lines of source code can be greater than the number of highlighted rows in the failure report as multiple lines of source code may be executed (and highlighted)

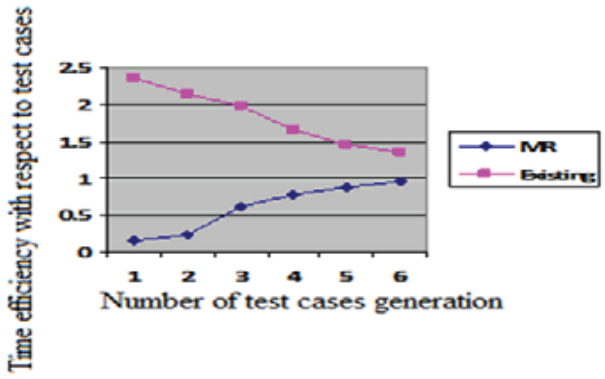

Figure 3. Experimental evaluation for test case process with respect to techniques.

between two rows in the failure report. We can go on to employ program slicing techniques to further exclude statements having no impact on the output value reported in row 78. In this way, the focus of debugging will be further narrowed down. Such a program slicing component, however, has not been incorporated into our current system owing to resource limitation, and also because this component is not the focus of our present research project.

As shown in the Figure 3 performs extra specifications in test case generation of program modulations with method declarations and method implementation and if statements for loops were processed in to program specifications. We find that, in spite of the fact that this rate is high, the highlighted lines are to be sure a bona fide reason impact chain: The deficiency in the system has brought about numerous extra cycles of a circle body and, in this manner, a not insignificant rundown of execution follows inside of the circle body have been highlighted. The quantity of source lines highlighted, then again, is entirely little (SLP = 4 %.) The second biggest FRP (26%) happens when form 12 is confirmed against MR4. This is additionally because of cycles of a circle. Truth be told, the 128 lines of highlighted execution follows include just 16 exceptional columns, and the quantity of source lines highlighted is additionally little (SLP = 5 %.)

6. Conclusion

In this paper we proposed an approach that alleviates the oracle problem for that we uses the metamorphic testing that generates the metamorphic relations to overcome the oracle problem. It generates more number of test cases when compare to the existing system their by it was easy for the tester to detect the faults in the program and the proposed approach uses the fuzzy logic to identify the degree of correctness. The usage and robotization of semi-demonstrating is likewise moderately less demanding than routine project demonstrating methods. This is on account of changeable relations are weaker than project accuracy and, henceforth, they can be less demanding to demonstrate.

7. References

1. Ahmed AAF, Shaheen M, Kosba E. Software Testing Suite Prioritization using Multi criteria Fitness Function. 22nd Conference on Computer Theory and Applications, ICCTA; 2012. p. 160–66.

2. Gotlieb A, Botella B. Automated Metamorphic Testing. 27th Annual International Proceedings Computer Software and Applications Conference, COMPSAC'03; 2003. p. 34–40.
3. Posonia AM, Jyothi VL. Context-based Classification of XML Documents in Feature Clustering. Indian Journal of Science and Technology. 2014 Jan; 7(9):1355–58.
4. Bharathi B, Kulanthaivel G. A Simple Method for Deriving LQN-models from Software-models Represented as UML Diagrams. Indian Journal of Science and Technology. 2012 Feb; 5(2):2148–54.
5. Malz C, Jazdi N, Gohner P. Prioritization of Test Cases using Software Agents and Fuzzy Logic. IEEE Fifth International Conference on Software Testing, Verification and Validation, (ICST); Montreal: QC. 2012. p. 483–86.
6. Tangeri D, Beszedes A, Gergely T, Vidacs L, Havas D. Beyond Code Coverage-an Approach for Test-Suite Assessment and Improvement. 2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops, (ICSTW); Graz. 2015. p. 1–7.
7. Jeffrey D, Gupta N. Improving Fault Detection Capability by Selectively Retaining Test Cases during Test Suite Reduction. IEEE Transactions on Software Engineering. 2007; 33(2):108–23.
8. Barr ET, Harman M, McMinn P, Shahbaz M, Yoo S. The Oracle Problem in Software Testing: A Survey. IEEE Transactions on Software Engineering. 2015; 41(5):507–25.
9. Wang F, Yao L-W, Wu J-H. Intelligent Test Oracle Construction for Reactive Systems without Explicit Specifications. IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing; Sydney: NSW. 2011. p. 89–96.
10. Lodha GM, Gaikwad RS. Search Based Software Testing with Genetic using Fitness Function. Innovative Applications of Computational Intelligence on Power, Energy and Controls with their impact on Humamity, CIPECH. Ghaziabad. 2014. p. 159–63.
11. Fraser G, Arcuri A. Evolutionary Generation of Whole Test Suites. 11th International Conference on Quality Software (QSIC); Madrid. 2011. p. 31–40.
12. Dong G, Wu S, Wang G, Guo T, Huang Y. Security Assurance With Metamorphic Testing and Genetic Algorithm. 2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology; Toronto: ON. 2010. p. 397–401.
13. Liu H, Kuo F-C, Towey D, Chen T Y. How Effectively Does Metamorphic Testing Alleviate The Oracle Problem?. IEEE Transactions on Software Engineering. 2014; 40(1):4–22.
14. Liu H, Liu X, Chen T Y. A New Method for Constructing Metamorphic Relations. 12th International Conference on Quality Software; Xian: Shaanxi. 2012. p. 59–68.
15. Jones JA, Harrold MJ. Test- Suite Reduction and prioritization for Modified Condition/ Decision Coverage. IEEE Transactions on software Engineering. 2003; 29(3):195–209.
16. Mayer J, Guderlei R. An Empirical Study on The Selection of Good Metamorphic Relations. 30th Annual International Computer Software and Applications Conference, COMPSAC'06; Chicago: IL. 2006; 1:475–84.
17. Staats M, Loyola P, Rothermel G. Oracle-Centric Test Case Prioritization. 2010 IEEE 23rd International Symposium on Software Reliability Engineering (ISSRE); Dallas: TX. 2012. p. 311–20.
18. Bashir MB, Nadeem A. A Fitness Function for Evolutionary Mutation Testing of Object Oriented Programs. IEEE 9th International Conference on Emerging Technologies, (ICET); Islamabad. 2013. p. 1– 6.
19. Julian Menezes R, Albert Mayan J, Breezely George M. Development of a Functionality Testing Tool for Windows Phones. Indian Journal of Science and Technology. 2015 Sep; 8(22):1–7.
20. Rehman Khan SU, Parizi RM, Elahi M. A Code Coverage-Based Test Suite Reduction and Prioritization Framework. Fourth World Congress on Information and Communication Technologies (WICT); Bandar Hilir. 2014. p. 229–34.
21. Segura S, Hierons RM, Benavides D, Ruiz-Cortes A. Automated Test Data Generation on the Analysis of Feature Models: A Metamorphic Testing Approach. Third International Conference on Software Testing, Verification, and validation (ICST); Paris. 2010. p. 35–44.
22. Chen TY, Kuo FC, Liu H, Wong WE. Code Coverage of Adaptive Random Testing. IEEE Transactions on Reliability. 2013; 62(1):226–37.
23. Kanewala U. Techniques for Automatic Detection of Metamorphic Relations.IEEE 7th International Conference on Software Testing, Verification, and Validation Workshops, (ICSTW); Cleveland:OH. 2014. p. 237–38.
24. Vineeta, Singhal A, Bansal A. A Study of Various Automated Test Oracle Methods. 5th International Conference on Confluence the Next Generation Information Technology Summit; Noida. 2014. p. 753–60.
25. Xue-ying MA, Sheng B-K, He Z-F, Ye C-Q. A Genetic Algorithm for Test- Suite Reduction. 2005 IEEE International Conference on Systems, Management and Cybernatics; 2005; 1:133–39.
26. Dong Y, Peng J. Automatic Generation of Software Test Cases based on Improved Genetic Algorithm. 2011 International Conference on Multimedia Technology (ICMT); Hangzhou.2011. p. 227–30.

AUTHOR QUERY

Au1: Please indicate correspondence author