COVERAGE BASED TEST SUITE AUGMENTATION TECHNIQUES-A SURVEY

Bharti Suri[1], Prabhneet Nayyar[2]
[1]Assistant Professor, USIT
Guru Gobind Singh Indraprastha University, Delhi, INDIA
bhartisuri@gmail.com
[2]M.Tech, USIT
Guru Gobind Singh Indraprastha University, Delhi, INDIA
prabhneetnayyar@gmail.com

ABSTRACT

Regression testing is the activity of retesting a program so as to ensure that no new errors have been introduced into the previously tested code. However this activity does not involve rerunning the entire test suite but selecting only a few test cases that exercise the changes. Suppose there is a program P and P' is it's modified version. The regression test suite so selected must be able to reveal the differences between P and P' that would help the developer discover errors caused by changes. More and more emphasis has been laid in identifying the regression test suites and ordering them. However, less focus have been laid on the effectiveness of regression test suite in response to changes. While performing regression testing we also need to check whether the existing test suites are sufficient for handling the changes that are introduced. If they are not adequate then providing guidance for creating the new test cases that would be targeting the changed behavior of the program. This problem is called as test suite augmentation. Many test suite augmentation techniques have been proposed in this regard. The main aim of this paper is to explain the concept of test suite augmentation and review the existing techniques based on coverage criteria.

KEYWORDS: *Test Suite Augmentation, Matrix, Directed Test Suite Augmentation*

1. INTRODUCTION

One of the fundamental activities during software evolution is regression testing. Regression testing [4] is the testing of the modified software to ensure that the changed parts of the software behave as it is intended to behave and to ensure that no new errors have been introduced as a result of these changes. This activity however does not involve rerunning the entire suite but selecting a subset of it that exercise the changes. More and more emphasis has been laid on majorly three activities namely: regression test selection, test suite reduction and test suite prioritization. Regression test selection basically identifies test cases that we do not need to rerun on the modified version of the software. Test suite reduction on the other hand takes criteria into consideration and accordingly eliminates redundant test cases in a test suite. Test suite prioritization [19] [20] is a concept related to finding the defects earlier by ordering test cases in a test suite. However little attention have been paid to determining whether the existing test suite adequately exercises the modified software and if not providing a suitable guidance for creating new test cases for the same. This problem is called as test suite augmentation problem. This research is divided into four sections. The first section describes about test suite augmentation. In the next section the techniques proposed for solving the test suite augmentation problem and mainly focuses on identifying the affected portions[2][3][4][12] .In the next section other techniques for test suite augmentation that mainly focuses on test case generation[24][25] along with augmentation algorithm [13] are discussed. A comparative analysis is also obtained after the techniques have been reviewed.

2. BACKGROUND

2.1 Test Suite Augmentation [7]

Consider a program P and let P' be its modified version. Let T be a test suite for P. Regression testing is concerned with validating P' and to facilitate this engineers often begin by reusing T. On the other hand test suite augmentation is not concerned with reusing the test suite rather concerned with two basic tasks. Firstly it is used to identify all the affected elements that are those portions of P' for which new test cases are needed. Secondly it is also concerned with creating or providing a suitable guidance or creating test cases that exercise these affected elements.

2.2 Augmentation Basics [13]

Test suite augmentation mainly consists of two main activities. The first activity is concerned with identifying all the affected portions .Second activity is concerned with creation of test cases for the affected elements. This section describes some of the augmentation basics that are needed to be considered while performing augmentation process. Following three factors are mainly considered.

- Coverage Criterion: Most augmentation techniques operate on specific code coverage criteria. The focus has been on branch coverage and it is more likely to scale to larger systems.
- Identifying Affected Elements: Test suite augmentation techniques involve identifying the affected elements .So this factor affects the augmentation process.
- Ordering Affected Elements: This factor also affects the augmentation process that is the order in which the affected elements are considered.

3. AUGMENTATION TECHNIQUES FOR IDENTIFYING AFFECTED ELEMENTS

This section discusses the augmentation techniques and lays more emphasis on identifying the affected portions and correspondingly providing guidance for generating the test requirements for those affected portions. However these techniques do not actually generate the test cases.

3.1 Matrix [1]

The augmentation technique proposed by Apiwattanapong et al MATRIX [1] stands for maintenance oriented testing requirement identifier and examiner. It is a technique that made use of dependence analysis [5] and partial symbolic execution [9][23] and a tool was also used that implemented it partially. The technique used the information obtained between the two versions of the program so as to identify the program elements affected by changes and accordingly created a set of testing requirements. However, the technique worked on a single change at a time .Another shortcoming of this technique was that it could not handle some of the program constructs like unbounded dynamically allocated objects. The algorithm proposed for finding out the testing requirements works as follows. In the first step it uses the data based on difference between the original program and the modified programs so as to identify the pairs of corresponding statements in them. The technique also made use of symbolic execution [9] so as to find the statements in original and modified programs that can be executed after the changes. In the next step it brings out a comparison in path conditions and symbolic states and generates testing requirements based on the changes. In order to identify the testing requirements the algorithm performs testing requirement identification that computes each statement and its counterpart in terms of their path conditions and symbolic states to identify test requirements. The algorithm was applied on nine versions of two Siemens programs Tcas and Schedule that generated the testing requirements. The information was then fed to the Matrix Analyzer to determine which testing requirements were satisfied by the test suite.

3.2 Augmentation Technique for Evolving Software [5]

Santelices et al [5] proposed an augmentation technique which was an improvement of Matrix technique as it identified all the change propagation paths. The technique also handles complex and multiple changes. It also provided with a tool that fully implemented the technique as compared to matrix[1] .The approach combines dependence analysis[5] and partial symbolic execution[9]. The technique is based on two main intuitions[5].The first intuition is that test criteria for changed software must require test cases to reach potentially affected areas of the code along different relevant paths therefore, approach requires that specific chains of data and control dependences be exercised. The second intuition is that test requirements for changed software must account for the state of the software and the effects of changes on that state. This approach limits its analysis to a given distance from the changes considered. The technique uses the concept of chain and state requirements. State requirements were the requirements that are generated to test single changes. Chain requirements were the requirements that take into consideration a particular dependence distance from the change. These requirements were used to find the output differencing between the two programs.

This approach performed four main steps. In the first step the approach computed all chains of dependences from the changes to statements at distance up to d from the changes. Next the approach evaluated certain conditions under the identified chains behave differently in P and P'. Thirdly both chains and conditions were evaluated as test requirements .It then assessed the extent to which T satisfied the identified requirements. The computed coverage indicated the adequacy of T with respect to the changes between P and P'. The algorithm to compute chain and state requirements evaluated the testing requirements for a set of changes. The inputs to the algorithm are the original and the modified programs ,a set of changes in modified programs and a parameter max used to identify the maximum length of the chain and a set of output statements in P'. The outputs are the chain and state requirements. The algorithm aligns the original and the modified programs so as to match all the dependence points. It also obtains a chain set having chains of length zero. It checks whether each and every chain in the chain set has reached the output or not. If the chain reached the output it was added as a requirement. All the chains that have maximum length were added to chain requirements. The affected portions were identified by utilizing forward direct dependence that identified the statements that were control dependent on input statement and data dependent on the definition of a variable at a dependence distance. For all the affected statements in the modified program the algorithm finds the corresponding matching statements in the original version. The testing requirements were generated for all the affected elements. The algorithm uses partial symbolic evaluation that is used to identify all the path conditions and symbolic states. The algorithm computes the state requirements so as to ensure the propagation of the infection at the end of each chain. The technique was applied on three programs namely Tcas, Nanoxml-v1, Nanoxml-v3 and the technique was found to be effective and scalable.

4. AUGMENTATION TECHNIQUES FOR TEST CASE GENERATION

This section explains the main augmentation algorithm and other techniques for test suite augmentation.

4.1 Main Augmentation Algorithm [6]

The main augmentation algorithm [6] initialized with a set of test cases and an ordered set of affected elements that were to be covered. The algorithm also specifies an iteration limit. Each branch to be covered was taken one by one from the ordered set and it was checked whether it had been covered or not by the test cases. If it was not covered then test case generation algorithms were called so as to generate new test cases. The algorithm is successful if new test cases were generated and these test cases were augmented with the original test cases.

4.2 Genetic Test Suite Augmentation [13]

Genetic algorithms [8] are widely used for structural test case generation [24] [25].The algorithm begins with an initial test data population and the population continuously evolves. Two parameters are defined in using genetic approach. Chromosome represents test inputs and a fitness function that defines how well a chromosome satisfies the intended goal. The main genetic algorithm evaluates all the chromosomes and selects a subset fittest to mate. These are then combined in the crossover stage where one half of the population is exchanged with other half of the population to generate new population and a small percentage is mutated to add diversity back into the population. Since augmentation technique used genetic algorithms for test case generation therefore the technique was named as genetic test suite augmentation [13]. The algorithm for genetic test suite augmentation [13] accepts four inputs: existing test suite T set of affected elements, uncovered branch b and an iteration limit. The population corresponds to the existing test cases. The algorithm was run on each branch in the set of affected elements. Initial test data population starts by selecting those test cases that reach uncovered branch b. An iteration limit was set that determines the number of generations to be repeated by that algorithm. The algorithm first computes a fitness of the population of the test cases which was done by executing all the test cases. In the next step ordering and selection of chromosomes (test data population) was done. The population is then divided into two halves in which first chromosome in the first half is mated with the first chromosome in the second half. Mutation is done so as to add diversity back into the population. All the test cases in the current population were then executed. The technique was applied to non trivial java application nano xml that has multiple versions. Augmentation was performed as the system goes through three iterations of evolution. The test cases so generated provided 74.7%, 83.6% and 78.5% of the branches in version1, version 2 and version 3 respectively.

4.3 Directed Test Suite Augmentation [7]

The technique worked by reuse of existing test suite [7] and an incremental concolic testing [11] approach. It presented an algorithm that operated in three main steps. A technique Dejavu [4] which was a regression test selection technique was used to identify the affected and unaffected test cases. The next step of the algorithm reruns all the affected test cases and accordingly a testing objective was calculated and returned a set of branches that need to be covered. The algorithm operates as follows. The algorithm selects a branch from a set of uncovered branch set. Next it locates all the branches in the branch set for which source node is a predicate node as these are the immediate targets for test generation. The branches are then ordered. For each branch with a source node the algorithm finds all the path conditions for test cases whose execution trace reach the source node. For each path condition the algorithm then deletes all the predicate nodes following the source node and then this source node was then negated so as to generate another path condition. If the path condition is not encountered before then a constraint solver [11] is called to generate test cases. The technique was applied to 42 versions of Tcas program and it provided a better branch coverage.

5. ANALYSIS AND DISCUSSION

This section brings out a comparative analysis of augmentation techniques in the form of a table based on some specified parameters.

Table1: Comparative analysis Of Test Suite Augmentation Techniques

Technique	**Matrix**	**Augmentation technique for evolving software**	**Directed test suite augmentation**	**Genetic test suite augmentation**
Proposed By	Tawees Apiwattanapong et al[1](2006)	Raul santelices et al[5](2008)	Zhihong Xu et al [7](2009)	Zhihong Xu et al [13](2010)
Technique To identify Affected Portions	Dependence analysis and symbolic execution	Dependence analysis and symbolic execution	Dejavu[4]	Not Applicable
Technique for Test generation	Not Applicable	Not Applicable	Incremental concolic testing	Genetic algorithm
Coverage	Path coverage	Path coverage	Branch coverage	Branch coverage
Tool used	Matrix using Java	Matrix reloaded using Java	Java	Sofya[26]
Advantages	Generated testing requirements for modified software	Effective and more scalable and cost effective	Relies on symbolic execution which is impractical	More flexible than directed test suite augmentation and generated good quality test cases
Disadvantages	Does not focuses on complex changes	Relies on symbolic execution which is impractical	Less Flexible	More expensive
Size of LOC	134 LOC	131,3497 and 4782 LOC for tcas,nanoxml v1 and v2 resp	200 LOC	7000LOC
Artifacts	Tcas and schedule programs	Tcas and nanoxml v1 and nanoxml v3	Applied to 42 versions of Tcas	Applied to three versions of nanoxml programs

		programs	programs	

The four techniques have been studied in detail. The analysis suggests that Apiwattanapong et al [1] and Santelices et al [5] proposed techniques in which more emphasis was given on path coverage. These techniques [1] [5] focused on single and multiple changes. The techniques proposed by Zhihong Xu et al [7] [13] focused on branch coverage. Genetic technique for augmentation was more flexible but expensive than the directed test suite augmentation technique. Genetic technique for augmentation also focused on generating good quality test cases.

6. RELATED WORK

There are many techniques that are related to test suite augmentation approach. These techniques are broadly divided into four categories. The first category takes into account the coverage of program entities namely statement, branches and definition use pairs [4] [14] [15] and defines testing criteria for the software. Techniques that fall into the second category generate testing requirements based on program modifications. Binkley [12][21] and Rothermel and Harrold [2] uses System Dependence Graph based slicing to generate testing requirements on the basis of data and control flow relations involving a change. Another technique based on slicing[21] proposed by Gupta and colleagues [3] overcame the costs associated with building system dependence graphs .The technique computes chains of data and control dependences from the change to the output statements. The third category of techniques produce requirements for fault based testing that also incorporates propagation conditions. RELAY framework given by Richardson and Thompson [16] computes a set of conditions to propagate the effects of faults to the output. A fault based testing given by Morell [17] uses symbolic evaluation to find out fault propagation equations. A fourth class of techniques usually adds existing test suites to improve their fault revealing capability [22]. Harder and colleagues [18] proposed a technique that was based on a model of the behavior of methods.

7. CONCLUSION

In this research various augmentation techniques based on coverage criteria have been studied. Based on review a table has been formulated which compares the techniques on different parameters. It was found that these techniques provided a suitable guidance for generating the test cases. Genetic and directed test suite augmentation techniques not only provided guidance but also created test cases.

REFERENCES

[1] T. Apiwattanapong, R. Santelices, P. K. Chittimalli, A. Orso, and M. J. Harrold. Matrix: Maintenance-oriented testing requirements identifier and examiner. In *Test.: Acad. Ind Conf. Pract. Res. Techn.*, pages 137–146, Aug. 2006.

[2] G. Rothermel and M. J. Harrold. Selecting tests and identifying test coverage requirements for modified software. In Int'l Symp. Softw. Test. Anal., Aug 1994.

[3]R. Gupta, M. Harrold, and M. Soffa. Program slicing-based regression testing techniques. J. Softw. Test., Verif. Rel., 6(2):83–111, June 1996.

[4] G. Rothermel and M. J. Harrold. A safe, efficient regression test selection technique. ACM Trans. Softw. Eng. Meth, 6(2):173–210, Apr. 1997.

[5] R. Santelices, P. K. Chittimalli, T. Apiwattanapong, A. Orso, and M. J.Harrold. Test-suite augmentation for evolving software. In Auto. Softw.Eng, Sept. 2008.

[6]Z. Xu and G. Rothermel. Directed test suite augmentation. In Asia-Pac.Softw. Eng. Conf., Dec.2009.

[7] S. Yoo and M. Harman. Test data augmentation: Generating new test data from existing test data. Technical Report TR-08-04, Dept. of ComputerScience, King's College London, July 2008.

[8] R. P. Pargas, M. J. Harrold, and R. R. Peck. Test-data generation using genetic algorithms. Softw. Test., Verif. Rel., 9:263–282, Sept. 1999

[9] S. Person, M. B. Dwyer, S. Elbaum, and C. S. P˘as˘areanu. Differential symbolic execution. In Int'l. Symp. Found. Softw. Eng., pages 226–237, Nov. 2008.

[10] Z. Xu, M. Cohen, and G. Rothermel. Factors affecting the use of genetic algorithms in test suite augmentation. In *Gen.Evol. Comp. Conf*, July 2010

[11K. Sen, D. Marinov, and G. Agha. CUTE: A concolic unit testing engine for C. In *Proc. Int'l Symp. Found. Softw. Eng*, pages 263–272, Sept. 2005.

[12] D. Binkley. Semantics guided regression test cost reduction.*IEEE Trans. Softw. Eng.*, 23(8), Aug.1997.

[13] Zhihong Xu, Yunho Kim, Moonzoo Kim, Gregg Rothermel, Myra B, Cohen, Directed Test suite Augementation: Techniques and TradeOffsACM November 7-11 2010

[14] P.Frankl and E.J. Weyuker An application family of data flow criteria. IEEE Trans on softw Eng...14(10):1483-1498, Oct 1988

[15] J.W Laski and B.Korel. A data flow oriented program testing strategy. IEEE Trans on softw.Eng. 9(3):347-354, May 1983

[16] D.Richardson and M.C Thompson The RELAY model of error detection and its application. In Proc of Workshop on Softw.Testing, Analysis and Verif. Pp223-230, July 1988

[17]L.Morell A Theory of Fault Based Testing, IEEE Transactions on Software Engineering, 16(8):844-847, August 1990

[18] M.Harder, J Mallen and M.D Ernst. Improving test suites via operational abstraction. In Proceedings of the 25th IEEE and ACM SIGSOFT International Conference on Software Engineering (ICSE 2003), pages 60-71, May 2003

[19] A Srivastava and J.Thiagarajan.Effectively prioritizing tests in development environment. In Proc of Int'l Symp.on Softw.Testing and Analysis, pp 97-106, July 2002

[20] G.Rothermel, R.Untch, C.Chu and M.Harrold .Test case Prioritization .IEEE Trans on softw Eng., 27(10):929-948, Oct .2001

[21]S. Horwitz, T.Reps, and D.Binkley Interprocedural slicing using dependence graphs.ACM Trans on Prog.Lang and Systems, 12(1):26-60, Jan 1990

[22] G. Rothermel, M.J Harrold, J.Ostrin and C.Hong.An empirical study of the effects of minimization on the fault detection capabilities of test suites. In Proceedings of the international Conference on Software Maintenance, pages 34-43, November 1998

[23] L.A Clarke and D.J Richardson .Applications of symbolic evaluations .Journal of systems and software, 5(1):15-35, February 1985

[24]H.Waeselynck, P.Thevenod-Fosse and O.Abdellatif-Kaddour,"Simulated annealing applied to test generation: Landscape characterization and stopping criteria,"Emp.Softw Eng, vol 12, no.1, pp.35-63, 2007

[25]G.Wasserman, D.Yu, A.Chander, D.Dhurjati, H.Inamura, and Z.SU,"Dynamic test input generation for web applications"in Prac Int'l Symp Softw Test Anal, July.2008, pp.249-260

[26] A. Kinneer, M. Dwyer, and G. Rothermel, "Sofya: A flexible framework for development of dynamic program analysis for Java software," University of Nebraska - Lincoln, Tech. Rep.TR-UNL-CSE-2006-0006, Apr.2006

Author Biographies:

Bharti Suri is an Asst. Professor at the University School of Information Technology, Guru Gobind Singh Indraprastha University, Kashmere Gate, Delhi, India. She holds master's degrees in computer science and information technology. She is pursuing PhD in the area of software testing. Her areas of interest are software engineering, software testing, software project management, software quality, and software metrics. She is a lifetime member of CSI. She has completed University Grants Commission (UGC) funded major research project as co-investigator in the area of software testing. She has many publications in national and international journals and conferences to her credit. She can be contacted by e-mail at bhartisuri@gmail.com.

Prabhneet Nayyar is currently pursuing master's degree (M.Tech) in Information Technology (IT) from the University School of Information Technology, Guru Gobind Singh Indraprastha University, Kashmere Gate, Delhi, India. She holds bachelor's degree (B.Tech) in Information technology (IT). Her areas of interest are software engineering, software testing and software development. She can be contacted by e-mail at prabhneetnayyar@gmail.com.