Integrated Intelligent Research (IIR)
International Journal of Computing Algorithm
Volume: 04 Issue: 01 June 2015 Pages:41-44
ISSN: 2278-2397

Test Suite Reduction Based on Fault Detection with Cost Optimization

B.Nevedha[1], N.Rajkumar[2]
[1]Dept. of Software Engineering, Sri Ramakrishna Engineering College, Coimbatore, India
[2]Professor, Dept. of Software Engineering, Sri Ramakrishna Engineering College, Coimbatore,India
Email: nevedha.b@gmail.com

Abstract-Test Suite Reduction is an optimization technique to identify the minimally sized subset of test cases with enforced constraints involved. The main purpose of test suite reduction is to deduce increased number of test cases that in turn increase the time and cost involved in execution. Fault Detection is the method of identifying the faults that affect the outcome of the system either logically or syntactically. This paper focuses on the reduction of the test suite that has high fault identification rates and also incurs low cost of execution of test cases. The proposed approach includes a new parameter Fault Detection Effectiveness to identify fault rates of test suite; an algorithm for test suite reduction based on priority of requirements; a low cost framework to identify the execution of test cases with minimum budget. Thus, the proposed work defines a test suite that has high fault detection effectiveness providing maximum coverage to requirements at minimum cost of execution.

Keywords-Fault Detection Effectiveness, Optimization, Coverage, Time of Execution, Cost Framework

I. INTRODUCTION

Software testing is the process of analyzing software to promote confidence that the actual behavior of the software correctly adheres to its specification of the requirements. Testing should systematically uncover different classes of errors in a minimum amount of time and with a minimum amount of effort. Secondary benefit of testing is that it demonstrates that the software appears to be working as stated in the specifications [2]. Testing helps is verifying and validating if the Software is working as it has to do the job. Methodologies to test the application may be either static or dynamic. It also provides a goal, unique perspective of the software to allow the organization to appreciate and understand the risks of software implementation. Test techniques include, both the process of executing a program or application with the intent of finding software bugs and also check if right application is created to satisfy the requirements [1].As software grows and evolves so does the accompanying test suites. More test cases will be required over time to test for new or changed functionality that has been introduced to the software, or to guard against a particular bug that has been previously discovered [7]. As time progresses, some test cases in a test suite will likely become redundant with respect to a particular coverage criterion, as the specific coverage requirements exercised by those redundant test cases are also exercised by other test cases in the suite. Notice that the property of a test case being redundant is relative to a specific set of coverage requirements.Software testing, depending on the testing method that is applied, and is dynamic throughout the development process. However, effort involved in testing is maximum after the requirements have been elicitated and the process of coding development has been completed.

II. RELATED WORK

As software grows and evolves, so too do the accompanying test suites. More test cases will be required over time to test for new or changed functionality that has been introduced to the software, or to guard against a particular bug that has been previously discovered. As time progresses, some test cases in a test suite will likely become redundant with respect to a particular coverage criterion, as the specific coverage requirements exercised by those redundant test cases are also exercised by other test cases in the suite [5]. Notice that the property of a test case being redundant is relative to a specific set of coverage requirements.For example, a test case exercising a certain set of statements A is redundant relative to the statement coverage of a test suite if the union B of all the statements exercised by the other test cases in the suite is such that $A \subseteq B$.However, that same test case may actually not be redundant relative to, for instance, definition-use pair coverage, if the test case exercises a unique definition-use pair that is not exercised by any other test case in the suite [1, 7]. It is important, therefore, to remember that redundancy of a test case is a property that is relative to some specific set of requirements.As test suites grow in size, they may become so large that it becomes desirable to reduce the sizes of the suites. This is especially true in situations where an extreme programming approach is followed which, among other guidelines, stresses the daily testing of software from the very first day of software development [3]. Test suite reduction is one general technique that has been proposed to address the problem of excessively large test suites.

III. PROBLEM DEFINITION

The aim of this paper is to reduce the size of the test suite by removing the unwanted and redundant test cases. The first Step is to generate the test cases and group them to corresponding test suites. The fault detection effectiveness of test suites is calculated and test suites with high value are taken for optimization. The test suites are then optimized based on their priority and coverage of requirements. A representative set of test cases are thus achieved. A parameter cost is incorporated to yield low cost test suite with high coverage. The characteristics of the test cases are then analyzed.

IV. PROPOSED APPROACH

The approach is defined in four major steps

International Journal of Computing Algorithm
Volume: 04 Issue: 01 June 2015 Pages:41-44
ISSN: 2278-2397

1. Generation of Test Cases
2. Grouping of test suites based on FDE
3. Optimization of Test Suite
4. Cost Calculation

A. *Generation of Test Cases*
The first step is to construct test data. All types of receivers, parameters, return values of target methods, constructors executed to produce the test case. A test case is defined as the sequence of call descriptions, the test case generation takes the possible data as a solution space to search, and apply approaches to find for a good solution. A flow graph for the program is generated and test cases covering the corresponding flow are documented. The test case of the corresponding program is inhibited and considered using the generate function of the build package.

B. *Grouping of Test Suites based on FDE*
The test cases are pooled together to form a test suite. The test cases are grouped together in the test suite based on the similarity existing among them taking into account their coverage criterion and their fault detection rate. The coverage criterion defines the coverage of test suite to the requirements. Thus the end result is the test suite with maximum fault detection effectiveness taken for optimization.
The steps include the following
1. Identify the test cases of the program.
2. Order the test cases to the test suite based on their coverage criterion and the type of faults identified.
3. Generate test suites containing relevant test cases of the program.
4. Determine the Fault Detection Effectiveness of the test suite.

FDE= (Number of faults identified by test suite) / (total number of faults in a program)

C. *Optimizing the Test Suite*
Test Suite Reduction technique returns the representative set that contains the optimized set of test cases. The process of execution in the reduction is organized as a sequence of activities which is described below in steps.
The Algorithm steps include
1. Initially, all requirements are unmarked.
2. For each requirement that is exercised by only one test case each, add each of these test cases to the representative set and mark the requirements covered by the selected test cases.
3. Consider the unmarked requirements in the increasing order of cardinality of the set of test cases exercising the requirements and add them to representative set.
4. If there is a tie among multiple test cases then choose the test case with high coverage of requirements.
5. If the tie is not broken even after the application of the constraint then randomly pick the test case.
6. The Representative Set is the optimized test suite with the test cases that yield maximum coverage.

The algorithm contains a helper function used to select the next test case to include in the reduced suite. The input to the algorithm is a mapping of each requirement covered by an original test suite to the set of test cases in the suite covering that particular requirement. The goal is to find a representative set of test cases, of smallest possible size, covering the same set of requirements as the original suite. The approach follows a heuristic to greedily select the test cases that cover the requirements that are the hardest to satisfy, until all requirements are covered.

D. *Cost Calculation*
The Algorithm gives the representative set with optimized set of test cases. The optimized set may sometimes incur a high cost of execution. Hence, the parameter cost is incorporated to obtain a low budget test suite with maximum coverage. Thus, it is possible to obtain a representative set of test cases which takes low cost for execution for testing and also that satisfy maximum number of requirements. Also the prioritization of the requirements is done so as to consider that the representative set covers the maximum prioritized requirements that are defined.The steps for cost calculation include

- Order the requirements based on the priority of the execution. Higher priority requirements are placed on the top and are necessary to be executed descending down to the next less priorities.
- Calculate the cost of the representative set.
- Cost = Coverage / Cardinality
- If there occurs a tie where two test suites have same minimum cost thenchoose the test suite that covers the high priority requirements.

V. FLOW GRAPH

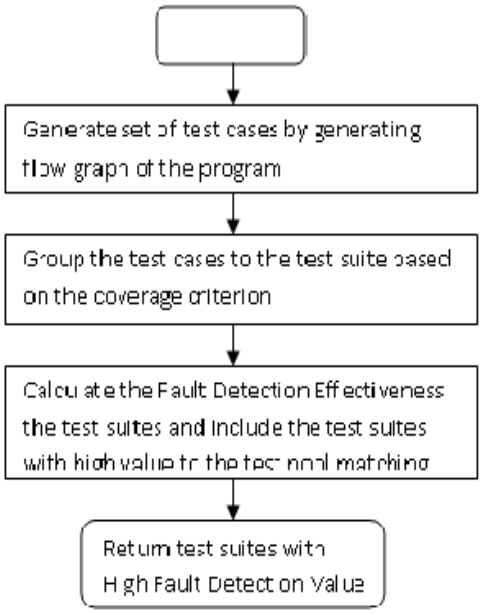

Fig 1: Generation of Test Suites for optimization

The flow explains the step by step process of execution of the entire process. The first flow chart describes the fault detection technique and formation of test suites. The second flow chart shows the optimization of tsuites based on the cardinality coverage of requirements.

International Journal of Computing Algorithm
Volume: 04 Issue: 01 June 2015 Pages:41-44
ISSN: 2278-2397

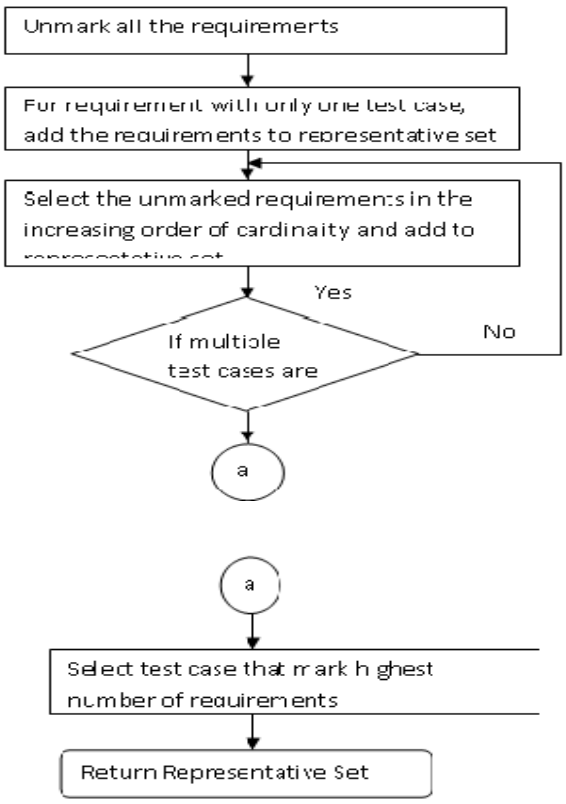

Fig 2: Representative Set of optimized Test Suites

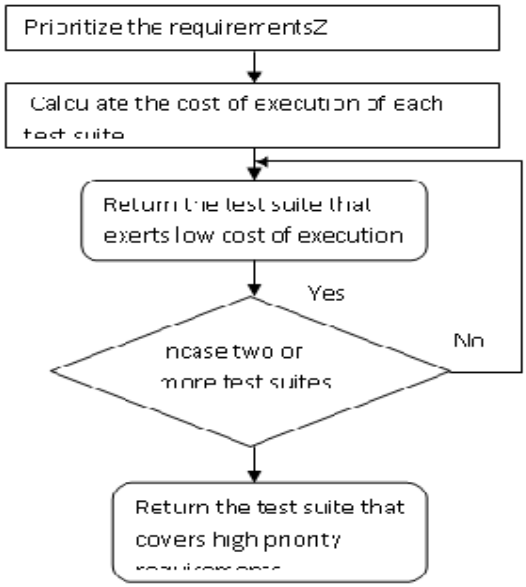

Fig 3: Choosing the Low Cost Test Suite

The flow explains the step by step process of execution of the entire process. The first flow chart describes the fault detection technique and formation of test suites. The second flow chart shows the optimization of the test suites based on the cardinality and coverage of requirements. The third flow chart illustrates the prioritization of the requirements and also the cost calculation. The end result is the low cost test suite with high coverage.

VI. EXPRIMENTAL RESULTS

Table 1: Input-Output table for each stage

STAGES	INPUT	OUTPUT
Generation of Test Cases	Any program that needs to be tested	Generation of Flow Graph and Derivation of test cases
Grouping of test suites based on FDE	Set of test cases derived	Grouping of test cases to test suites and forming test pool with high FDE test suites
Identifying the Optimized test suites	Set of test Suites	Representative set of optimized test case
Cost Calculation	Optimized test suites	Low cost test suite covering maximum requirements

VII. PERFORMANCE ANALYSIS

The results of proposed algorithm thus obtained by adapting FDE and cost framework with prioritization of requirements for test suite reduction shows a considerable decrease in the cost and time of execution when compared to the random reduction of the test cases from the test suite. The method is simple to implementand also shows a greater efficiency in the process of test suite reduction. It also focuses on the primary parameters such as budget and time which are beneficial for the users to evaluate the system at minimum cost at a small deadline. A requirement A is deemed harder to satisfy than a requirement B if A is covered by fewer test cases (has a smaller associated test case set size) than B.

VIII. CONCLUSION

This paper provides a novel approach to the Test Suite Reduction methodology. This approach uses a parameter Fault Detection Effectiveness to choose the test suites with high fault identification rates. Test Suites thus chosen are reduced based on their coverage of requirements and also their effectiveness to cover the prioritized requirements. Cost framework developed helps to identify test suites executing at minimum budget. Results show that by using this approach the cost and time involved in execution of the test cases is greatly reduced and also the test suites highly efficient in detecting the faults existing in the system.

IX. FUTURE WORK

In future, many other metrics like fault tolerance, accuracy, risk management and time of execution parameters can be included to the algorithm making it more effective. The work can also be extended by creating a fault log table, categorizing the faults that occur with the program and taking the necessary

International Journal of Computing Algorithm
Volume: 04 Issue: 01 June 2015 Pages:41-44
ISSN: 2278-2397

actions that help in increasing the precision of the system taken.

REFERENCES

[1] C.T.Lin, K.W.Tang, G.M.Kapfhammer, "Test suite reduction methods decrease regression testing costs by identifying irreplaceable tests", Information and Software Technology, May 2014.

[2] H.Zhong, L.Zhang,H.Mei, "An experimental study of four typical test suite reduction techniques", Information Software Technology. Vol 50, 2008, pp. 534–546.

[3] J.Harrold, R.Gupta, M.L.Soffa, "A methodology for controlling the size of a test suite", ACM Transaction, Software Engineering Methodology, Vol 3, 1993, pp. 270–285.

[4] D.Jeffrey, N.Gupta, "Improving fault detection capability by selectively retaining test cases during test suite reduction", IEEE Transaction, Software Engineering, Vol 33, 2007, pp.108–123.

[5] J.W.Lin, C.Y.Huang."Analysis of test suite reduction with enhanced tiebreaking techniques", Information Software Technology, Vol 51, 2009, pp.679–690.

[6] T.Y.Chen, M.F.Lau, "A new heuristic for test suite reduction", Information Software Technology, Vol 40, 2008, pp. 347–354.

[7] T.Y.Chen, M.F.Lau, "A simulation study on some heuristics for test suite reduction", Information Software Technology, Vol 40, 2009, pp. 777–787.

[8] DeMillo, A.P.Mathur, "On the Use of Software Artifacts to Evaluate the Effectiveness of Mutation Analysis" Technical Report SERC-TR-92-P, Purdue University, West Lafayette, 2000.

[9] Do.H.Elbaum., G.Rothermel, "Supporting controlled experimentation with testing techniques: an infrastructure and its potential impact", Empirical Software Engineering, Vol10, 2005, pp.405–435.

[10] P.G.Frankl, R.G.Hamlet, B.Littlewood, L.Strigini, "Evaluating testing methods by delivered reliability", IEEE Transactions Software Engineering, Vol24, pp.586–601, 1998.

[11] J.A.Jones, M.A.Harrold,"Test-suite reduction and prioritization formodified condition/decision coverage", IEEE Transaction, Software Engineering, Vol. 29,2003, pp.195–209.

[12] D.Binkley, "Semantics guided regression test cost reduction" IEEE Transaction Software Engineering Vol 23, 1997, pp. 498–516.