International Journal of Application or Innovation in Engineering & Management (IJAIEM)

Web Site: www.ijaiem.org Email: editor@ijaiem.org

Volume 3, Issue 8, August 2014 ISSN 2319 - 4847

An Empirical study of Statement Coverage Criteria to reduce the test cases-A Review

Sudesh Kumar[1] , Nupur Gupta[2]

[1]Department of Computer Science
BRCM college of Engineering and Technology, Bahal

[2] Department of Computer Science
BRCM college of Engineering and Technology, Bahal

ABSTRACT

Software testing is too much important phase of software development life cycle and it is very expensive. Software testing is particularly expensive for developers of high-assurance software, such as software that is produced for commercial systems. It is well known that software testing is an important activity to ensure software quality because software quality is increasingly important factor in software marketing. When we test or retest the software then development organizations always desire to validate the software from different views. But exhaustive testing requires program execution with all combinations of values for program variables, which is impractical due to resource constraints. Test cases can be generated automatically for some of the applications by various testing techniques. But the number of test cases are very large and we have to reduce the test cases to test the software efficiently. Test case reduction method reduces the test cases that is not necessary for testing the software. In this paper, we will study Black box testing technique to generate the test cases and Statement coverage criteria for the reduction of test cases that will reduce time and cost spent on testing.

Keywords: software testing, test suite reduction, representative set.

1. INTRODUCTION

Software testing is an important but expensive phase of Software Development Life Cycle (SDLC). Exhaustive testing provides more confidence about the quality and reliability of the developed software during maintenance phase. In this technique, Test manager executes the programs with all infinite combinations of values for program variables [1]. Generally the domain of a program is infinite and cannot be used as a test data. Exhaustive testing requires every statement in the program and every possible path combination to be executed at least once. According to Rothermel et.Al. the product of about 20,000 lines of code requires seven weeks to run its entire test case [2]. A test case is defined in IEEE standard as: "A set of test inputs, execution, and expected results developed for a particular objective,such as to exercise a particular program path or to verify compliance with a specific requirement"[3]. While a test suite is a collection of test script or test cases that are used for validating bug fixes (or finding new bugs) within a logical or physical area of product. Test suite minimization is an optimization problem with the following goal: To find a minimally-sized subset of the test cases in a suite that exercises the same set of coverage requirements as the original suite. The key idea behind minimization techniques is to remove the test cases in a suite that have become redundant in the suite with respect to the coverage of some particular set of program requirements. The minimization problem can be formally stated as follows: The Test Suite Minimization Problem Given: a set (test suite) T of candidate test cases t1, t2, ..., tn and some set of coverage requirements R, where each test case covers a set of software requirements r1, r2, ..., rn, respectively, such that r1 [?] r2 [?] ... [?] rn = R .

Problem:

find a minimally-sized subset of test cases T′ [?] T, comprised of tests t'^{1}, t'^{2}, ..., t′m, each test covering a set of software requirements r′1, $r'2$, ..., r′m , respectively, such that r'^{1} [?] r'^{2} [?] ……. [?] r′m = R The test suite minimization problem is an instance of the more general set-cover problem, which when given as input a collection S of sets, each set covering a particular group of entities, is to find a minimally-sized subset of S providing the same amount of entity coverage as the original set S. It is often the case that software testers are subject to time and resource constraints when testing software. Due to such constraints being present for software retesting every time the software is modified, it is important to develop techniques that keep test suite sizes manageable for testers. When a collection of test suites becomes very large, a tester may not have enough time or resources available to test the software using every test case in each suite. In such a situation, the tester has no choice but to run fewer test cases to stay within the allowed time and resource constraints. The problem for the tester is then to decide which test cases are the most important and should therefore be run. This is where test suite minimization techniques become helpful.For practical purpose, we require test case selection strategy that can be easily adopted without heavy overhead or need of sophisticated tool support [4]. Generally software are tested through a test case. Large number of test suite is generated using automated tools. But the real problem is the selection of subset of test cases and/or high order test cases for validates the System Under Test (SUT). A test case Reduction (TCR) technique helps test manager to find out representative set of test cases at little cost. By doing so, we can reduce the test case execution, management, and storage cost [5]. In this paper, we propose a novel test reduction technique that selects test cases based on their statement-coverage therefore weight. Weight refers to the number of occurrences of a particular test case that cover different statement of the program under test. In This technique, first weight of all generated test cases is calculated. Next test cases with higher weight are selected and marked its entire corresponding requirement as satisfied. In case of test cases having same weight random selection strategy is used.

2.TEST CASE GENERATION

The test case generation results in a collection of test cases called a test suite. The generation may be done manually or automatically. After the test suite is produced, a test harness executes the test suite against the implementation under test. This produces a test result, which is compared to the expected result, prescribed by the specification, by a test oracle.. Ideally, the verdict of a test should be pass or fail. If all generated tests pass, then this shows conformance between the test and the specification. A failed test is a system failure, i.e., the system does not deliver the expected result (erroneous or with incorrect timing). If the test is carried out under the specified circumstances, then the failure shows that the system has an error, i.e., a design flaw.If a test has failed, the system (as a whole), does not conform to the test. The test itself might not conform to the specification, and in that case the test case should be changed and not the system. Further, the specification may not express the intention of the system. In this case the specification may be changed and the test cases rewritten.Often the expected test result can be incorporated into the test cases so that the test harness can make the verdict itself; in this case the oracle is a part of the test harness. This is especially good if the test cases consist of long sequences, because the test harness can stop further interaction and execute the next test case if it discovers an error.

3.METHODOLOGY

Our test case generation approach consists of four parts such as:

3.1 I/O Specification
3.2 Test Case Generation
3.3 Statement Coverage Requirement
3.4 Optimal Test Case Evaluation

3.1 I/O Specification

I/O Specification shows the Input and Output of the projected program in detail. This document is used as primary source of input for test case generation.

3.2 Test case generation

After the specification of input, for each input we produce the test cases. These test cases are produced by some testing technique.

3.3 Statement coverage requirements

Statement coverage specify the test cases that covers all the statement of program

3.4 Optimal test case evaluation

This is done by our procedure of weight assignment method. Outputs of this method produce the optimal test cases.

The techniques available for test suite reduction have their own Pros and Cons.

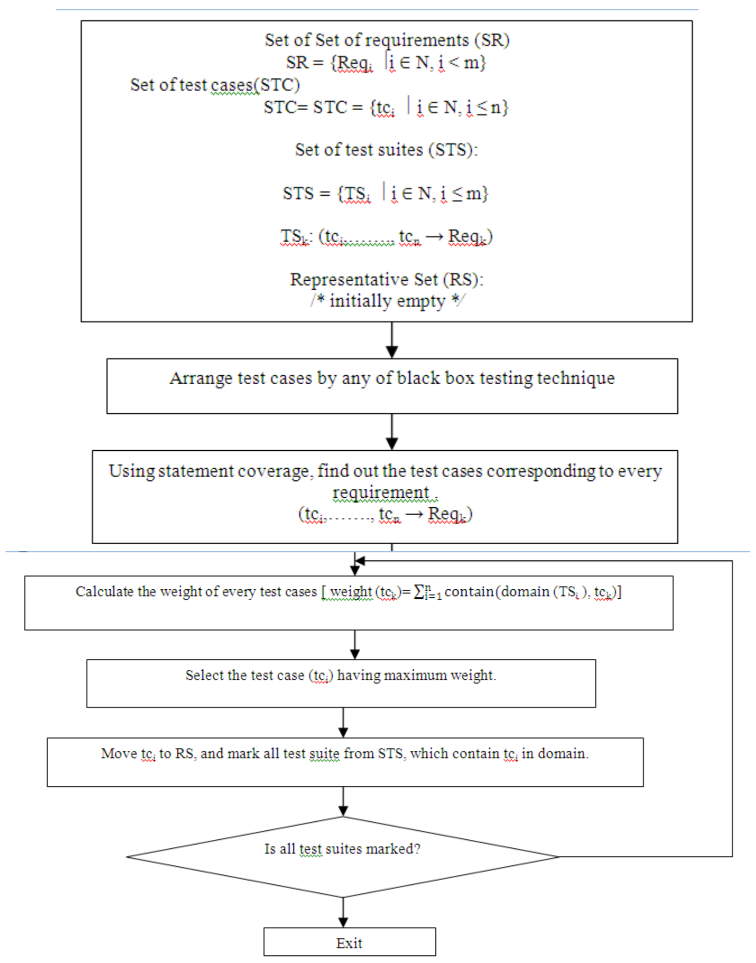
The main advantages of using TCR techniques are,

(1) That test manager spend less time in executing test cases,
(2) Spend less time in examining test results, and
(3) Spend less time in managing the data associated with testing.

A potential drawback of existing minimization approaches is that they may significantly decrease the fault detecting capability. This is because, these techniques

(1) Permanently remove some of the test cases from the original test suite.
(2) It may leads to deduction in the fault detection capability of the reduced suite .

Thus, the Trade off between the times required to execute and reduced test suites and their fault detection capability must be considered when applying test suite reduction techniques. Rothermel et. al. conduct some experimental studies to investigate this risk. They found that this loss is so minimal as compared to the benefits that test manager can get by using these techniques [8,9]. In literature, a number of test case reduction heuristics have been presented by different researchers [7,10,11,12].Harrold et. al. presents a heuristic which selects test case that hits most of the requirements that are hardest to satisfy [6]. In case of tie, it picks the test case that hit most of the requirements that are next hardest to satisfy and if it is still a tie, one of the tests is chosen at random. Ronne presents a new test reduction heuristic called as Multi-Hit Minimization algorithm [3]. This heuristic first determines different probabilities known as PIE (Propagation, Infection, and Execution) analysis for statement's sensitivity.


Set of Set of requirements (SR)
SR = {Req_i | i ∈ N, i < m}
Set of test cases(STC)
STC= STC = {tc_i | i ∈ N, i ≤ n}
Set of test suites (STS):
STS = {TS_i | i ∈ N, i ≤ m}
TS_k: (tc_i,........, tc_n → Req_k)
Representative Set (RS):
/* initially empty */
Arrange test cases by any of black box testing technique
Using statement coverage, find out the test cases corresponding to every requirement .
(tc_i,......., tc_n → Req_k)
Calculate the weight of every test cases [weight (tc_k)= Σ_{i=1}^{n} contain(domain (TS_i), tc_k)]
Select the test case (tc_i) having maximum weight.
Move tc_i to RS, and mark all test suite from STS, which contain tc_i in domain.
Is all test suites marked?
Exit

4. STEPS OF TECHNIQUE

The reduction technique requires an association between the test cases and the testing requirements of the program. This technique identifies optimal test cases and selects the non-redundant test cases based on their weights. In this study, we are interested in determining that at what percent it reduces the number of test cases? The procedure is as follows:

Step1

Inputs:

Set of requirements (SR):

$$SR = \{Req_i \mid i \in N, i < m\}$$

Set of test cases (STC): each test case completely satisfies one or more requirements.

$$STC = \{tc_i \mid i \in N, i \leq n\}$$

Set of test suites (STS):

$$STS = \{TS_i \mid i \in N, i \leq m\}$$

Where each test suite in the set of test suites is a function from one or more test cases to exactly one requirement. i.e., TS_k: (tc_i,……., $tc_n \rightarrow Req_k$) TS_k means that each test case tc_i,..., tc_n, in the test suites satisfies the requirement Req_k. Output: Representative Set (RS): /* initially empty */

Step 2: Arrange the test cases by any of testing technique i.e. boundary value analysis, Robustness testing, worst-case testing technique.

Step3: Using statement coverage, find out the test cases corresponding to every requirement .

(tc_i,……., $tc_n \rightarrow Req_k$)

Step4: Calculate the weight of every test case. Where weight of a test case is the number of its occurances in set of test suite. The weight of a test case tc_k is:

$$\text{weight}(tc_k) = \sum_{i=1}^{n} \text{contain}(\text{domain}(TS_i), tc_k)$$

Step5: Select the test case (tc_i) having maximum weight. In case of a tie between test cases, use random selection.
Step6: Move tc_i to RS, and mark all test suite from STS, which contain tc_i in domain. If all test suite of STS are marked then exit, otherwise go back to step 4.
Step7: Finally, we get optimal number of test cases.
We can also find the percentage of reduction in test cases by using:
% of reduced test cases

$$= \frac{\text{No. of test cases get reduced}*100}{\text{Total no. of test cases}}$$

5. CONCLUSION

In this paper , we have discussed about test case reduction by using weight criteria and Statement Coverage Method . We have seen that we have reduced the number of test cases by using weight criteria. The result of our experimental study shows that we get maximum reduction in test cases by using this method. This reduction helps test manager in achieving nearly exhaustive testing level testing. Simultaneously it reduces the size of test suites by eliminating unnecessary test cases. This method also reduces the test case storage, management. Ultimately, it is beneficial to optimize time and cost spent on testing. In our experimental study, our approach consistently performed better on average than other test suite minimization techniques. This method is also applicable to object oriented languages like Java, VB, C++.

REFERENCES

[1.] Saif-ur-rehman khan, Aamer Nadeem,A;"Testfilter:A statement-coverage based test case reduction technique",in IEEE multitopic conference(inmc' 06),pp.275-280,December 2006.
[2.] G. Rothermel, R.H. Untch, M.J. Harrold, "Prioritizing Test Cases for Regression Testing", In IEEE Transactions on Software Engineering (TSE'01), Vol. 27, No. 10, pp. 929-948, October 2001.
[3.] J.V. Ronne, "Test Suite Minimization: An Empirical Investigation", Bachelors Thesis, June 1999. Retrieved from URL:http://www.ics.uci.edu/jronne/pubs/jvronne-uhc-thesis.pdf.
[4.] Zhi Quan Zhou, Arnaldo Sinaga, Lei Zhao, Willy Susilo, Kai-Yuan Cai , "Improved Software Testing Cost-Effectiveness Through Dynamic Paritioning", In Proceedings of IEEE 9th International Conference on quality software,2009
[5.] Piyusha Tyagi, Sheetal K. Jain, Ravi Shankar Singhal,"The Enhanced Apporach for Test Suite Reduction", In Proceedings of International Conference On Issues and Challenges in Networking, Intelligence and Computing Technologies,Sept 2011.
[6.] M.J. Harrold, R. Gupta, M.L. Soffa, "A Methodology for Controlling the Size of Test Suite", In ACM Transactions on Software Engineering and Methodology (TOSEM'93), NY USA, pp. 270-285, 1993.
[7.] W.E. Wong, J.R. Horgan, A.P. Mathur, A. Pasquini, "Test Set Size Minimization and Fault Detection Effectiveness: A Case Study in a Space Application", In Proceedings of IEEE 21s' International Conference on Computer Software and Applications Conference (COMPSAC '97), 1997.
[8.] G. Rothermel, M.J. Harrold, J. Ostrin, C. Hong, "An Empirical Study of the Effects of Minimization on the Fault Detection Capabilities of Test Suites", In Proceedings of IEEE International Test Conference on Software Maintenance (ITCSM'98), Washington D.C., pp. 34-43. November, 1998.
[9.] G. Rothermel, M.J. Harrold, J.V. Ronne, C. Hong, "Empirical Studies of Test Suite Reduction", In Journal of Software Testing, Verification, and Reliability, Vol. 12, No. 4, December 2002.
[10.] A. Kandel, P. Saraph, M. Last, "Test Cases Generation and Reduction by Automated Input-Output Analysis", In Proceedings of 2003 IEEE International Conference on Systems, Man and Cybernetics (ICSMC'3), Washington, D.C., October 5-8, 2003.
[11.] B. Vaysburg, L.H. Tahat, B. Korel, "Dependence Analysis in Reduction of Requirement Based Test Suites", In Proceedings of the 2002 ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA'02), Roma Italy, pp. 107-111, 2002.
[12.] J.A. Jones, M.J. Harrold, "Test-Suite Reduction and Prioritization for Modified Condition/Decision Coverage", In IEEE Transactions on Software Engineering (TSE'03), Vol. 29, No. 3, pp. 195-209, March 2003.