Integrated Intelligent Research (IIR) International Journal of Computing Algorithm
Volume: 07, Issue: 01, June 2018, Page No.38-44
ISSN: 2278-2397

A Study on Coverage Criteria Based Test Case Reduction Techniques

P. Prema
Assistant Professor, Department of Computer Science,
Thiruvalluvar University Constituent Arts and Science College, Kallakurichi, Tamilnadu, India
Email: mrgprem@gmail.com

Abstract - Nowadays, software is an intrinsic part of human life and is everywhere such as media players, mobile phones, airways, etc., and is important that, these systems should perform their intended functions, as software failures may be expected at any where any point of time. Software testing is essential for developing software and one of the important phases is test case generation. As software size grows, test cases generated for them are also more in number. Even though many techniques are available for test case reduction, still there is a need for new techniques. This paper presents detailed description of software testing techniques and presents an overview of coverage criteria based test case reduction techniques, which helps new researchers to know about existing test case reduction techniques and come up with a new test case reduction technique.

Keywords - Coverage Criteria, Software Testing, Test case reduction techniques, Testing Techniques

I. Introduction

One of the essential phases in software development life cycle is software testing and it consumes more than 50% of total software development cost [5, 8, 15, 17, 21, 22, 23, 36, 40]. According to IEEE standards [18], “Software Testing is a process of analyzing a software item to detect the differences between existing and required conditions and to evaluate the features of the software item”. Myers [23] says that, software testing is a difficult task to perform exhaustive testing as the domain of program input is usually very large and can have many possible inputs. Dijkstra [12] says that “Testing can be used to show the *presence* of error, but never to show their absence!”.

There are three different phases in software testing: test case generation, test execution and test evaluation. In these three phases, test case generation is the core of any testing process [1, 22]. A test case is variables or a set of conditions or test input data and the expected results, under which a tester will determine whether a system works correctly or a system under test satisfies requirements. The test data is a set of input values to the program, which may be generated from the code or usually derived from program specifications. Expected results are determined with the help of program specifications. A test suite is a finite set of test cases which is used to test a program.

Test case reduction is one of the essential strategies among the testing community. The main idea of test case reduction techniques is to eliminate the test cases in a test suite that have become redundant with respect to the coverage of any particular set of system requirements. Section II provides a detailed description of software testing techniques, Section III presents literature review of coverage criteria based test case reduction techniques and finally conclusion is given in section IV.

II. Classification of Test case generation TECHNIQUES

Test case generation techniques are classified into: White Box Testing (WBT) and Black Box Testing (BBT) [6, 19, 23, 29, 38, 41]. Before applying any testing methods, the tester should understand the following testing principles [11]

- Once the requirement stage is complete testing should be planned
- All tests must be traceable to customer requirements
- Apply Pareto principle to software testing

- Exhaustive testing is not possible
- Independent testing team should conduct the test

A. White Box Testing (WBT)

In WBT, test cases are generated from a program itself and tester chooses inputs to execute paths through the program code and determines the appropriate outputs (see Figure 2.1). WBT is also called structural testing techniques and some of the WBT techniques (see Figure 2.2) are: control-flow based testing, data-flow based testing, mutation testing etc., which is used for test case generation [35, 43].

a) In control flow based testing, test cases are selected based on program's control flow. Statement coverage, branch coverage, condition coverage and path coverage are some of the control flow based testing techniques.

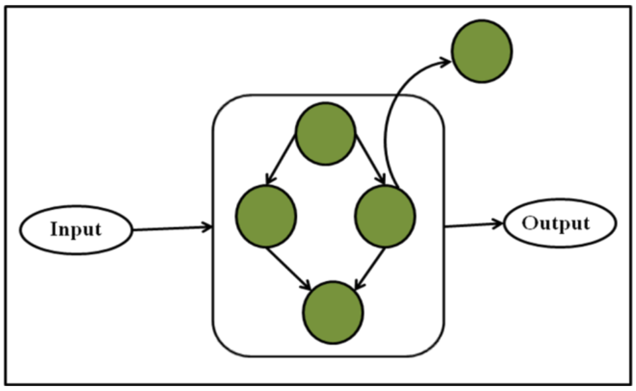

Figure 1: White Box Testing

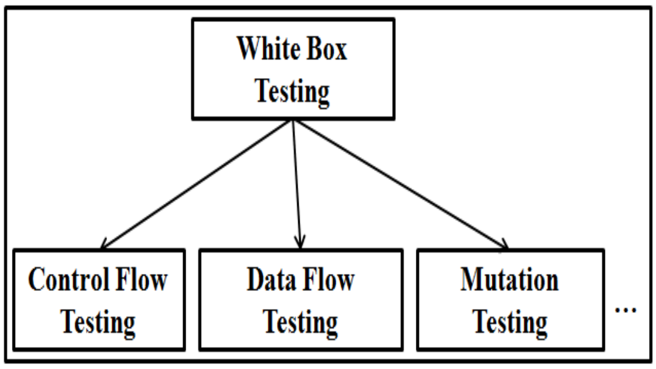

Figure 2: WBT Techniques

b) Data flow based testing techniques to explore the events which are related to the status of variables during the program's execution. The assignments of value and the uses of value are necessary events (that is, where the variables are defined and where they are used). Some of the data flow testing techniques are: all-definitions, all-c-uses (c means computation), all-p-uses (p means predicate) and all-du-paths (du means definition-use pair). However, these techniques are quite theoretical and complex to use in practice.

B. Black Box Testing (BBT)

Beizer [7] says that, BBT is an important technique for test case generation where test cases are generated without any knowledge of the internal structure of the software under investigation (see Figure 2.3). The tester should understand and specify what the desired output should be for a given input into the program. BBT is also called functional or specification based testing. In software development, the specification and system requirements exist prior to program implementation and the tester checks whether the application behaves exactly the way it is supposed to do which is based on its functional requirements. Jorgensen [19], Vergilio et al.[42], and Murnane et al. [27] say that, the specification of the program is used to generate test cases in BBT and the main advantage of specification based testing is that, they are independent of how the program is implemented, so the test cases will not be affected if the implementation changes.

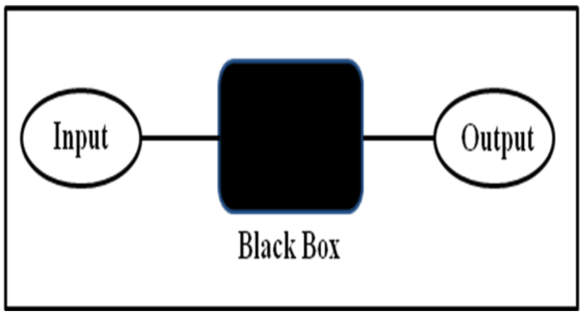

Figure 3: Black-Box Testing

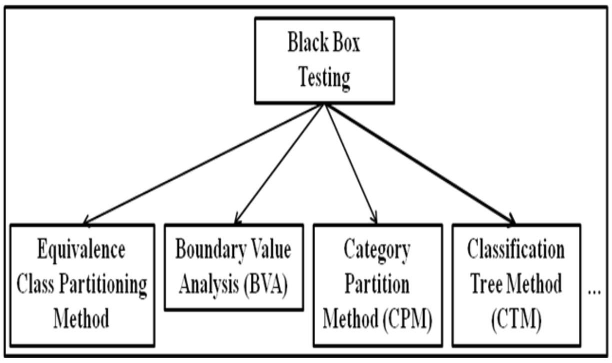
Black Box Testing
Equivalence Class Partitioning Method
Boundary Value Analysis (BVA)
Category Partition Method (CPM)
Classification Tree Method (CTM)
...

Figure 4: BBT Techniques

BBT can be classified into various categories **[**7, 24**]** and some of the categories are given in Figure 2.4. This section describes some of the BBT in detail.

a) Equivalence Class Partitioning Method (ECPM)

ECPM partitions the input domain into a relatively small number of subsets of input domains and select any one value from each partition as test data. Equivalence classes are defined using some of these guidelines which are given in [23, 24, 28, 41] (see Figure 2.5):

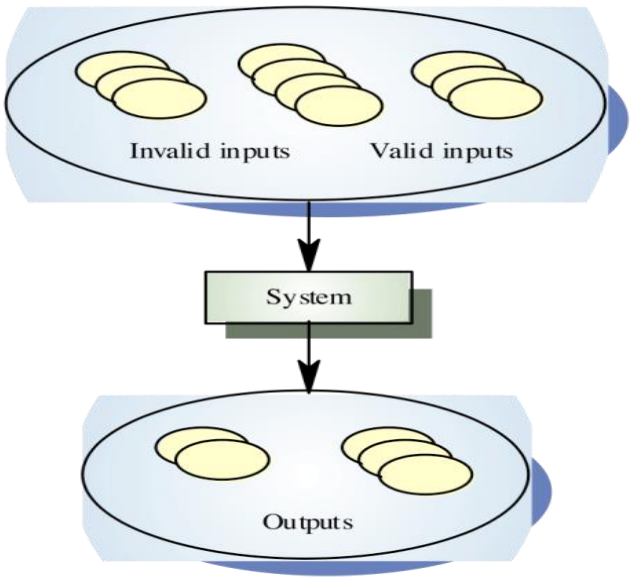

Figure 5: Equivalence Class Partition Method

- If the input specifies a range of valid values, define one valid class with values inside the range and define two invalid classes with values outside the range.
- If the input specifies the number (*N*) of valid values, define one valid and two invalid classes (none, and more than *N*).
- If the input specifies a set of valid values, define one valid class (within the set) and one invalid class (outside the set).
- If the program handles each valid input differently, then define one valid class for each valid input.
- If the input specifies a 'must be' situation, define one valid class and one invalid class. After identifying the test conditions, generate one test case for all valid input conditions and generate separate test cases for each invalid input.

b) Boundary Value Analysis (BVA) Technique

BVA is performed by creating tests at the extreme ends of an input domain in the system. More application errors occur at the boundaries of an input domain and the BVA technique is used to identify errors at boundaries rather than finding those exist in the center of the input domain. BVA follows multi-dimensional partitions of input domain rather than focusing on input conditions; it also focuses on output conditions. The following are guidelines for selecting test cases using BVA techniques [7, 10]

- Select the exact boundaries of the input domain
- Select value just below the extreme edges
- Select value just above the extreme edges

This technique is efficient only for variables with fixed values.

c) Category Partition Method (CPM)

Ostrand and Balcer [34], and Amla and Ammann [4] described a way to apply CPM to generate test cases based on formal specifications. In this CPM method, the tester can decompose the specification into the functional unit which is to be tested separately using Test Specification Language (TSL). TSL is a concise, easily modified representation of tests that can be understood by programmers, testers and managers. Offutt and Irvine [30] defined criterion for generating test cases for object-oriented software using CPM. The following are guidelines for generating test cases using CPM [24]:

(i) Identify the parameters and environmental conditions which affect the behavior of each functional unit.
(ii) Identify choices for each parameter and environment individually.
(iii) Determine constraints among the choices.
(iv) Generate all combinations, so called test frames, of parameter choices that satisfy the constraints
(v) Transform the test frames into test cases by instantiating the choices.

This method does not automatically validate the results of the test parameters to determine correctness with respect to the functional specifications.

d) Classification Tree Method (CTM)

Among the BBT categories, CTM is one of the widely used specification based testing techniques. Based on CPM, Grochtmann and Grimm [13] introduced CTM for deriving test cases from the functional specifications. They defined classifications as the different criteria for partitioning the input domain of the program to be tested [14], and classes as the disjointed subsets of values for each classification. Based on the specification, classification tree organizes the classification and classes into a hierarchical structure. For example, a program that calculates the sum of the square roots of two real numbers called A and B. The numbers A and B are two possible classifications and each of them has three classes "< 0", "= 0", and "> 0".

III. COVEERAGE CRITERIA BASED TEST CASE REDUCTION TECHNIQUES

Test case reduction techniques are used to eliminate the redundant test cases in a test suite that have become redundant with respect to the coverage of any particular set of system requirements. The test case reduction can be formally stated by Harrold et.al., [16] "Given a test suite T represents a set of test cases {tc_1, tc_2, …tc_n}, a set of coverage requirements X = {X_1, X_2, …X_n} and subsets of T, Y – {Y_1, Y_2, …Y_n}, where each test set is associated with X_i. The main objective is to find the representative subset XY $\subseteq$ Y that satisfies all of requirements and has at least one test case for each requirement X.”

Test case reduction techniques are used to reduce the testing cost and are broadly discussed by Rothermel et al. [37]. A framework for minimization of test suites is investigated by Agarwal [2]. An empirical study of test suite reduction was suggested by Rothermel et al. [36, 39]. Cichos and Heinze [9] proposed an efficient test suite reduction by merging pairs of suitable test cases. Mahapatra et al. [25] say that, minimum number of test runs, helps to improve test performance. Wong et al. [44] have presented an empirical study conducted to evaluate the effect of reducing the size of the test suite. Also a comparative study of test suite reduction is dealt by Zhong et al. [45]. Test suite reduction for regression testing of simple interaction between two software modules is proposed by Kichigin [20]. Heuristics for reducing the size of the test suite is presented by Harrold et al. [16]. Oliveria Neto et al. [33] have provided a technique for reducing the number of redundant test cases by considering the degree of similarity between test cases as the main factor for test case selection.

A coverage criterion plays a vital role in the test case reduction and many coverage criteria are there to measure the coverage of tests. Offutt et al. [1, 32] say that, coverage criteria describe a finite subset of tests out of the infinite or large number of tests, which measure the percentage of requirements that are satisfied. McMaster and Memon [26] have introduced a new coverage criterion for a test suite reduction based on the set of unique call stacks. Test suite reduction techniques and their classifications are proposed in [3].

In terms of state-based specification, different test criteria are defined by Offutt et al. [31]: (i) Transition coverage criteria (ii) Full predicate coverage criterion (iii) Transition pair coverage criterion (iv) Complete sequence criterion. An evaluation of three specification coverage criteria was performed by Abdurazik et al. [1]. Decision/condition coverage criteria have been proposed by Chilenski and Miller [8]. Zhu et al. [46] provides a comprehensive survey of existing test coverage criteria.

IV. CONCLUSION

When systems are in large size, test cases generated for them also have more in number. There is a need to reduce the number of test cases. There are many techniques available for test case reduction and still there is a need for new techniques to reduce the number of test cases. To help researchers, this paper provided a study of coverage criteria based test case reduction techniques.

References

[1] Abdurazik, A., P. Ammann, W. Ding and J. Offutt (2000) Evaluation of three specification-based coverage testing criteria. In Proc. of ICECCS 2000: 6th IEEE International Conference on Engineering of Complex Computer Systems, 179-187.
[2] Agarwal, H (1999) Efficient coverage testing using global dominator graphs. In Proc. of the 1999 ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering, 11-20.
[3] Alian, M., D. Suleiman and A. Shaout (2016) Test case reduction techniques – survey. International Journal of Advanced Computer Science and Applications, 7(5), 264-275.
[4] Amla, N. and P. Ammann (1992) Using Z Specifications in Category-Partition Testing in Systems Integrity, Software Safety, and Process Security: Building the Right System Right. In Proc. of the 7th Annual IEEE Conference on Computer Assurance (COMPASS '92), IEEE Computer Society, Los Alamitos, California, 3-10.
[5] Bertolino, A. (2007) Software Testing Research: Achievements, Challenges, Dreams. In Proc. of the Future of Software Engineering Symposium (FOSE’07), IEEE Computer Society, 85-103.
[6] BEIZER, B Software testing techniques, 2nd Edition. Van No strand Reinhold Co., New York, USA, 1990.
[7] BEIZER, B Black-box testing, Wiley, 1995.
[8] Chilenski, J. J. and S. P. Miller (1994) Applicability of Modified Condition/Decision Coverage to Software Testing. Software Engineering Journal, 193-200.

[9] Cichos, H. and T. S. Heinze (2011) Efficient Test Suite Reduction by Merging Pairs of Suitable Test Cases. In Proc. of the 2010 International Conference on Models in Software Engineering, LNCS 6627, 244-258.

[10] COPELAND, L. A practitioner's guide to software test design. Artech House Publisher, 2004.

[11] DAVIS, A. Principles of software development, McGraw-Hill, 1995.

[12] DIJKSTRA, E. W. Notes on structured programming. Technological University Eindhoven T. H. Report 70-WSK-03, 2nd Edition, 1970b.

[13] Grochtmann, M. and K. Grimm (1993) Classification Trees for Partition Testing. Software Testing, Verification and Reliability, 3(2), 63-82.

[14] Grochtmann, M., J. Wegener and K. Grimm (1995) Test case design using classification trees and the classification-tree editor CTE. In Proc. of the 8th International Software Quality Week (QW '95), Software Research Institute, San Francisco, California, 1-11.

[15] Harrold, M. J. (2008) Testing evolving software: current practice and future promise. In Proc. of the 1st India Software Engineering Conference (ISEC'08). 3-4.

[16] Harrold, M. J., R. Gupta and M. L. Soffa (1993) A methodology for controlling the size of a test suite. ACM Transactions on Software Engineering and Methodology (TOSEM)2, 2(3), 270–285.

[17] HETZEL, B. The complete guide to software testing, 2nd Edition. QED Information Sciences, 1988.

[18] IEEE (1990) Standard Glossary of Software Engineering Terminology: IEEE Standard 610.12-1990. ISBN 1-55937-067-X.

[19] JORGENSEN, P. C. Software testing: A craftsman's approach. CRC Press, 1995.

[20] Kichigin, D. (2010) A method for test suite reduction for regression testing of interaction between two software modules. LNCS, Springer – verlag, 177-184.

[21] Korel, B. (1990) Automated software test data generation. IEEE Transactions on Software Engineering, 16(8), 870-879.

[22] Ma, X., B. Sheng and C. Ye (2005) Test-suite reduction using genetic algorithm. Advanced Parallel Processing Technologies, Lecture Notes in Computer Science, 3756, 253-262.

[23] MYERS, G. J. The art of software testing. John Wiley & Sons, Inc., New York, 1979,2004.

[24] MATHUR, A. P. Foundations of software testing: fundamental algorithms and techniques, 1st Edition, Delhi, Pearson Education, 2008.

[25] Mahapatra, R. P., M. Mohan and A. Kulothungan (2011) Effective tool for test case execution time reduction. In Proc. of CSIT, IACSIT Press, Singapore, 1, 105-111.

[26] McMaster, S. and A. Memon (2005) Call Stacks Coverage for Test Suite Reduction. In Proc. of 21st IEEE International Conference on Software Maintenance, 539-548.

[27] Murnane, T. and K. Reed (2001) On the Effectiveness of Mutation Analysis as a Black Box Testing Technique. In Proc. of Software Engineering Conference, 12-20.

[28] Murnane, T., K. Reed and R. Hall (2007) On the learnability of two representative of equivalence partitioning and boundary value analysis. In proc. of Software Engineering Conference, ASWEC 2007. 18th Australian, 274-283.

[29] Nidhra, S. and J. Dondeti (2012) Black Box and White Box Testing Techniques - A Literature Review. International Journal of Embedded Systems and Applications, 2(2), 29-50.

[30] Offutt, A. J. and A. Irvine (1995) Testing Object-Oriented Software using the Category-Partition Method. In Proc. of the 17th International Conference on Technology of Object-Oriented Languages and Systems (TOOLS 17), 293 - 304.

[31] Offutt, A. J., Y. Xiong and S. Liu (1999) Criteria for generating specification-based tests. In Proc. of the Fifth IEEE International conference on Engineering of Complex Computer Systems, USA, 119-129.

[32] Offutt, J., S. Liu, A. Abdurazik and P. Ammann (2003) Generating test data from state-based specifications. Software Testing Verification and Reliability, 13, 25-53.

[33] Oliveira Neto, F. G., E. G. Cartaxo, P. D. L. Machado and J. F. S. Ouriques (2008) Reducing the size of test cases based on similarities. 2nd Brazilian Workshop on Systematic and Automated Software Testing, 44-53.

[34] Ostrand, T. J. and M. J. Balcer (1988) The Category Partition Method for Specifying and Generating Functional Tests. Communications of the ACM, 31(6), 676-686.

[35] OULD, M. A. and C. E. UNWIN. Testing in software development. The British Computer Society Monographs in Informatics. Cambridge University Press, Cambridge, England, 1986.

[36] ROPER, M. Software testing. McGraw Hill, 1994

[37] Rothermel, G., M. J. Harrold, J. Ostrin, and C. Hong (1998) An empirical study of the effects of minimization on the fault detection capabilities of test suites. International Conference of Software Maintenance, Bethesda, Maryland, 34-43.

[38] Rothermel, G., M. J. Harrold, J. Von Ronne, C. Hong and J. Ostrin (1999) Experiments to assess the cost-benefits of test-suite reduction. Technical Report 99-60-09, Computer Science Department, Oregon State University.

[39] Rothermel, G., M. J. Harrold, J. Von Ronne and C. Hong (2002) Empirical studies of test-suite reduction. Journal of Software Testing, Verification and Reliability, 12(4), 219-249.

[40] Saha, G. K. (2008) Understanding Software Testing Concepts. Ubiquity, 9(6), SOMMERVILLE, I. Software engineering, 9th Edition. Addison Wesley, 2004, 2010.

[41] Vergilio, S. R., J. C. Maldonado and M. Jino (1997) Constraint based selection of test sets to satisfy structural software testing criteria. In Proc. of the International Conference of the Chilean Computer Science Society, 256-263.

[42] Weiser, M. D., J. D. Gannon and P. R. McMullin (1985) Comparison of Structural test coverage metrics. IEEE Software, 2(2), 80-85.

[43] Wong, W.E., J. R. Horgan. A. P. Mathur and A. Pasquini (1999) Test set size minimization and fault detection effectiveness: A case study in a space application. Journal of Systems and Software, 48(2), 79–89. URL http://dx.doi.org/10.1016/S0164-1212(99)00048-5.

[44] Zhong, H., L. Zhang and H. Mei (2006) An experimental comparison of four test suite reduction techniques. In Proc. of the 28th International Conference on Software Engineering, ACM: New York, USA, 636–640.

[45] Zhu, H., P. Hall and J. May (1997) Software unit test coverage and adequacy. ACM Computing Surveys, 29(4), 366-427.