Volume 3, Issue 6, June 2013 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Prioritize Test Case Survey for Component-Based Software Testing

Karambir
Assistant Professor
UIET Department of Computer Sc. & Engg,
Kurukshetra University Kurukshetra, Haryana, INDIA

Rajni Rani
Research Scholar
UIET Department of Computer Sc. & Engg,
Kurukshetra University Kurukshetra, Haryana, INDIA

Abstract: Prioritizing test cases are one of the most significant tasks in the software testing process. Test case prioritization techniques schedule the execution of test cases in a organize manner such that to achieve an objective function with greater efficiency. In test case prioritization technique the highest priority test cases are executed first and then lowest priority thus makes the testing effective. Test case prioritization will help to improve the rate of fault detection. In this paper presented the overview of test case prioritization which specified coverage based test case prioritization and cost oriented test case prioritization techniques then measuring effectiveness of the various prioritization techniques. The effectiveness of prioritization techniques measures with the help of APFD (rate of fault detection) i.e. average percentage of faults detected per minute, impact of fault and cost of each component to be tested.

Keywords: Software Testing, Test case, faults, test case prioritization, Average percentage of Fault Detection (APFD), Average percentage of Fault Detection with cost (APFDc).

I. Introduction

Testing is a major aspect of software development. Software testing is verify and validate that the software that has been built to meet these specifications. Testing make sure that what you get at the ending is what you required. When developing a component-based software system, we test each component individually. Components may have been developed by different people, written in different programming languages and executed in different operating platforms. Therefore the interfacing among components needs to be tested. Unit and module testing is the testing of program units and modules independently from the rest of the program. Integration testing refers to testing interfaces between units and modules to assure communication between components is correct [1]. This is dissimilar to system testing where the entire integrated system is test as a whole. A test criterion is a collection of rules that guide to test requirements, or specific components in a program that must be covered during software testing. The Test suites are measured by the percent of requirements that they satisfy. Scheduling and Prioritizing test cases are one of the most significant tasks in the software testing process. Giving priority to test cases helps you identify which test cases (high priority) to pick up first while performing component testing and what test cases (low priority) can be left out. Test case prioritization will help to improve the rate of fault detection and testing provides faster feedback to developers. Also, during time crunches and while meeting hard deadlines it is practically impossible to execute all test cases, so during that phase of project it is crucial to identify the major areas of the application to be tested. And having the test cases prioritized helps plan the execution efficiently and effectively and at the same time meeting deadlines become more practical and achievable. Furthermore, authors[2],[3]stated that test case prioritization methods and process are required, because: There are some reasons that specified the needs of prioritization methods: 1) prioritization specify which test cases needs to run first, 2) without prioritization waste a lots of time and cost to run the entire test suite. Test case prioritization schedule the execution of test cases in a organize manner so that increases the efficiency of software testing. Large numbers of test cases are designed for test software products effectively. But it is not possible to execute every test case due to fix time limitation and some other reasons for instance availability of experts. Due to these reasons, there is need of reducing the number of test cases takes during the testing process. Prioritizing test cases helps to reduce the number of test cases. In this paper presents an approach to prioritizing the test cases Proposed approach prioritizes test cases basis on the risk and cost factors that can occur in a software project. In this paper section 2 presents the literature survey on test case prioritization. In section 3and 4 describes an overview of test case prioritization and conclusion.

II. Related Works

Gregg Rothermel *et al.* [4] described a number of techniques for prioritized test cases of regression testing, these are: 1) Techniques that ordered test cases based on total coverage of the code components, 2) Techniques that ordered test cases based on their total coverage of code components that not earlier covered, 3) Techniques that ordered test cases based on the estimated ability to disclose faults in code components that they covered. They reported the results of numerous experiments in which they had applied those techniques to a number of test suites for different programs and also

measured the fault detection rates that achieved by prioritized test suites, compared those rates that achieved by randomly ordered and optimally ordered test suites. The data analysis had shown that every test prioritization techniques studied to improve the fault detection rate of test suites and that improvement occurred during fault detection rate even with the less expensive of those techniques. Their results suggested that those techniques could improve the fault detection rate of test suites and that result occurred even for the least expensive techniques. Sebastian Elbaum *et al.* [5] presented the use of test case prioritization techniques in regression testing and focus on the objective of improving fault detection rate, they had addressed several additional questions raised by that work: 1) Can prioritization techniques be effective when targeted at specific modified versions, 2) what coarse granularity prioritization techniques and trade-offs exist between fine granularity, 3) can incorporation of measures of fault proneness into test case prioritization techniques improve their test effectiveness? To address these questions, they had performed several new controlled experiments and case studies. The data and analysis indicate that version-specific test case prioritization could produce statistically significant improvements in the rate of fault detection of test suites. To further complicated matters, both their controlled studies and case studies suggest that the relative effectiveness of prioritization techniques could vary across programs. Their case studies illustrated that, for specific programs and modification patterns, it was possible for some techniques not to outperform random and techniques that outperform random might vary, their controlled and case studies show that the "best" technique to use might vary across programs.

Hyunsook Do *et al.* [6] presented an empirical study of prioritization techniques that applied across four different Java programs to and provided JUnit test suites to those programs. Although different studies of test case prioritization had been conducted earlier, those studies focused on a few specific types of test suites and single procedural language, C. Their study in contrast, applied prioritization techniques to an object-oriented language tested by using the JUnit testing framework, to examine whether the results find out in previous studies generalize to other programming language and testing paradigms. Their results of effectiveness of prioritization techniques verify several earlier findings also revealing some differences regarding prioritization technique granularity effects and test suite granularity effects. These differences could be explained in relation to characteristics of the Java language and JUnit testing. Sebastian Elbaum *et al.* [7] proposed two strategies: First, the basic instance-and-threshold strategy, recommended the technique that has been successful in the largest proportion of instances in the past, accounting for cost-benefit thresholds. Second, the enhanced instance-and-threshold strategy, adds into consideration the attributes of a particular testing scenario, using metrics to characterize scenarios, and employing classification trees to improve the likelihood of recommending the proper technique for each particular case. They had assumed that the prioritization techniques examined have equivalent costs. For the relatively simple techniques they had considered, all operating at the level of function coverage and using binary "diff" decisions that could be retrieved from configuration management, this assumption seems reasonable. When seeking to extend these comparisons to other classes of techniques, however, this assumption would be less reasonable. Techniques that incorporate test cost or module criticality information, present different cost-benefits tradeoffs. These tradeoffs could be modelled and related to cost-benefit thresholds, allowing comparisons of differing-cost techniques, but this approach needs to be investigated empirically.

Hema Srikanth *et al.* [8] proposed a prioritization strategy called PORT (Prioritization of Requirements for Testing Version 1.1) by discovering three prioritization factors (PFs): 1) customer-assigned priority based on requirements, 2) the requirement complexity,3) volatility of requirements. Their preliminary set of research objectives was listed below:

G1: To identify the most important faults/failures earlier in system test.

G2: To increase software field quality.

G3: To create the minimal set of PORT prioritization factors that can be used to effectively for TCP.

The prioritization factors (PFs), the customer-assigned priority (CP) on requirements, requirements complexity (RC) and requirements volatility (RV) were assigned values. Customer-assigned priority was the value (1 to 10) that assigned by customer and that was based on the importance of the requirement. Requirements complexity was the value that assigned by the developer based on observed implementation difficulties of the requirement. Requirements volatility was the how many times a requirement has changed. Higher factor values indicate the requirement for prioritization of test case.

Praveen Ranjan Srivastava [9] proposed an algorithm for test case prioritization in order to improve regression testing. Analysis was done for the prioritized and non-prioritized cases with help of average percentage fault detection (APFD) metric. Graphs proved that prioritized case was more effective. The aim of this paper was to develop a test case prioritization technique that prioritizes test cases based on the detection of fault rate. Bo Jiang *et al.* [10] proposed the first family of adaptive random test case prioritization techniques and conduct an experiment to evaluate its performance. It explored the ART prioritization techniques with different test set distance definitions at different code coverage levels rather than spreading test cases as evenly and early as possible over the input domain. The empirical results show that their techniques were significantly more effective than random ordering. Moreover, the ART-br-max min prioritization technique was a good candidate for practical use because it could be as efficient and statistically as effective as traditional coverage-based prioritization techniques in revealing failures. Siripong Roongruangsuwan *et al.* [11] introduced a new "4C" type of test case prioritization techniques, which were: 1) customer requirement-based techniques, 2) coverage-based techniques, 3) cost effective-based techniques and 4) chronographic history-based techniques. First, the customer requirement-based techniques were methods that prioritize test cases according to customer requirement specifications. Second, the coverage based techniques were structural white-box testing techniques. They compared test program behavior against apparent intention of source code. This different with black-box testing, which compares test program behavior against a software requirement specification and also examined how the program works and its possible

drawbacks in structure and logic. Third, the cost effective-based techniques were methods for test cases prioritization on the basis of cost factors, for example cost of analysis and prioritization cost. Last, the chronographic history-based techniques were methods that prioritize test cases based on the test execution history factor.

Sangeeta Sabharwal *et al.* [12] proposed a GA (Genetic Algorithm) based approach that finding the test path that must be tested first. Scenarios and test paths were derived from state chart and activity diagram respectively. The proposed approach makes use of Information Flow model and genetic algorithm to find the path that to be tested first. They had proposed a technique for prioritization of test case scenarios by using the concept of basic IF (information flow) metric.

Sahil Gupta *et al.* [13] proposed an approach for test case prioritization in order to improve regression testing. Analysis was done for prioritized and non-prioritized test cases with the help of average percentage fault detection (APFD) metric. It was proven that when the prioritized cases were run then result was more efficient. They introduce two criterions on the basis of which they determined a function calls and statement factor which was used to prioritize the test cases.

III. Test Case Prioritization Overview

A. Test case generation

Test case is a combination of inputs, executing function, expected outputs. We have used the JUnit framework [6] for executing unit tests. JUnit test cases are Java classes it contains test methods that are grouped into test suites.

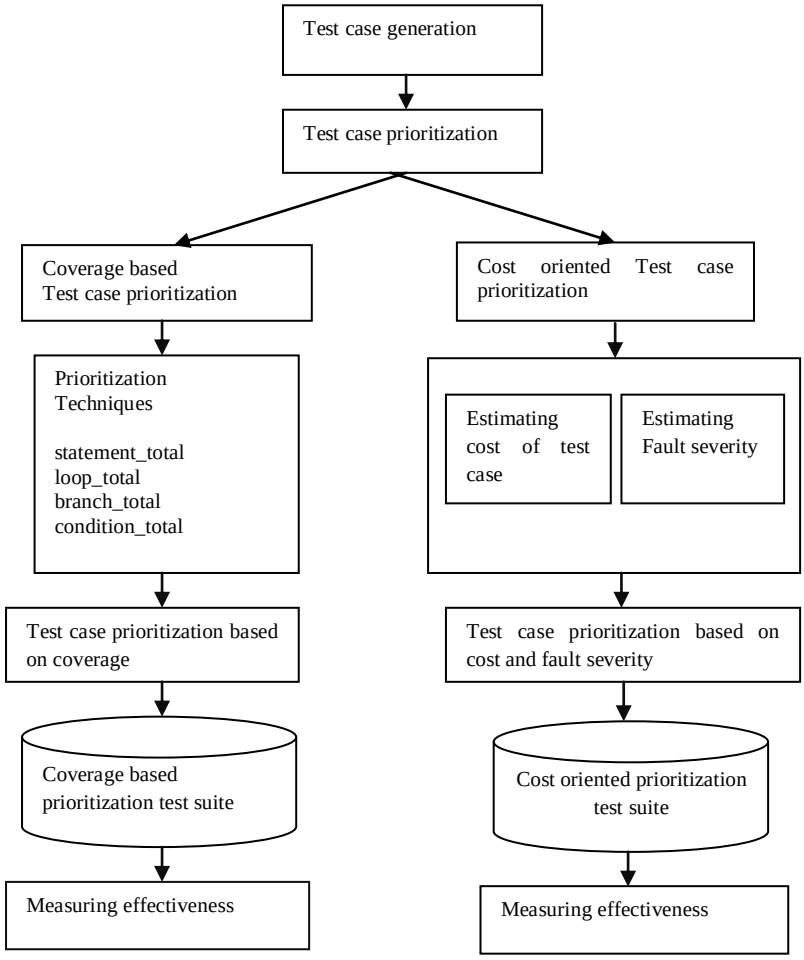

Figure1: Overview of Test Case Prioritization [14]

B. Test case prioritization

Test case prioritization is increase the software testing process efficiency as compared to the execution of test cases in non-prioritized order. Software is test by executing various test cases under different environment. But in case where all test cases are executed for same type of modules changed or unchanged, it results in redundancy. Test case prioritizations schedule test cases those results is increase in the performance of component testing. It is ineffective to repetitively execute every test case for every program function. So, test cases are prioritized in a queue for execution based on priority depending upon different principles and parameters.

1) Coverage based Test case prioritization ([15],[16], [17],[18]):

The various prioritization techniques are implemented based on code coverage information:

TABLE 1
Coverage based test case prioritization Techniques

Coverage-Based Prioritization Techniques	Description
Stmt_total prioritization	Stmt_total prioritization technique prioritizes test cases according to the total number of statements they cover.
Loop_total prioritization	It depends on coverage measured in terms of numbers of loops executed.
Branch_total prioritization	It depends on coverage measured in terms of numbers of branches executed.
Condition_total prioritization	It depends on coverage measured in terms of numbers of basic Boolean terms executed.

2) Cost oriented Test case prioritization ([19],[20],[21]):
In cost oriented test case prioritizations we need to calculate cost of prioritizing each test case. In this section we firstly estimate the cost of every test case and then estimate fault severity.

- *Estimating the test case cost*

We have estimated cost by measuring the actual time required to execute each test case i.e. machine or human time and monetary costs of test case execution. To calculate risk, cost and impact of component, we use following formula:
Risk of component (Rc)
Cost of component (Cc)
Impact of component (Ic) = Rc*Cc /100
This study proposes practical factors to prioritize and schedule test cases. There are need to calculate different prioritization factors, as follow:

TABLE 2
Proposed prioritization factor

Factor	Description
Cost factors[22]	
Execution cost (EC)	It is a total cost of running a set of test cases.
Cost of analysis (CA)	It includes source code analysis cost, analysis of changes between old and new versions and collection of execution of traces.
Cost of data preparation (CoDP)	It is a total cost of preparing all input values for test cases.
Cost of validation (CoV)	It is a total cost of validating the expected result and actual result.
Risk factors	
Risk occurred (RO)	An indicator of how many test cases have caused defects after execution.
Risk Severity (RS)	A dimension for classifying the seriousness for defects. It contains: critical and minor severity.
Impact[23]	
Test impact (TI)	It is an impact of test cases on rest component of the testing process. This evaluates how importance of test cases if it not to be executed.

- *Estimating fault severity*

There are two possible ways to estimate fault severity: assessing module criticality and test criticality. Module criticality assigns fault severity based on the importance of the module and test criticality is assigned directly to test cases based on an estimate of the importance of the test, or the severity of the faults each test case may detect. We need to generate faults and performing mutation testing for estimating the fault severity:
Fault generation: Fault is generated by locating naturally occurring faults and by seeding faults case. Faults occurring offer external validity, but they are expensive to locate and cannot be found in numbers sufficient to support controlled experimentation [24].
Mutation testing: Mutation testing is the process of mutating some segment of code and then this mutated code with some test data. If the test data is able to detect the mutations in the code, then the test data is quite good, otherwise we must focus on the quality of test data.

C. Measuring Effectiveness of the various prioritization techniques
In prioritization the effectiveness ([21],[24],[25]) of coverage based prioritization techniques using APSC, APBC, APLC and APCC metrices and for cost oriented prioritization techniques using APFDc metric.

1) APSC (Average Percentage Statement Coverage): It measures the rate at which the prioritized test suite covers statements [25].

2) APBC (Average Percentage Branch Coverage): It measures rate at which the prioritized test suite covers branches. APBC is also represented as Average Percentage Block Coverage [25].

3) APLC (Average Percentage Loop Coverage): APLC measures rate at which the prioritized test suite covers loops.

4) APCC (Average Percentage Condition Coverage): It measures the rate at which the prioritized test suite covers the conditions.

5) *APFD (Average Percentage of Fault Detection):* APFD ([5],[24]) measure that how quickly faults are found for a given test suites. APFD value range takes from 0 to 100. Higher the range of value shows that faster the rate of fault detection. For Average Percentage of Fault Detection we use following formula:

APFD=1-(TF1+TF2+_____+TFm)/nm+1/2n

6) APFDc (Average Percentage Faults Detected with cost): It measures the rate at which cost to be taken to detect defect.

IV. Conclusion

In this paper performed cost based prioritization techniques and examined the effectiveness of prioritization techniques in terms of rate of fault detection for some standard java programs. In this study it is shown that cost based prioritization techniques improved the rate of fault detection. The cost effective-based techniques were methods for test cases prioritization on the basis of cost factors, for example cost of analysis and prioritization cost. The survey of empirical results has shown that these techniques are significantly more effective than random ordering techniques. It can be as efficient and statistically as effective as traditional coverage-based prioritization techniques in revealing failures.

References

[1] B. Beizer, "*Software Testing Techniques*". Van Nostrand Reinhold, Inc, New York NY, 2nd edition, ISBN 0-442-20672-0, 1990.

[2] B. Korel and J. Laski, "Algorithmic Software fault localization",Annual Hawaii International Conference on System Sciences, pp. 246-252, 1991.

[3] Dennis Jeffrey and Neelam Gupta, "*Test Case Prioritization Using Relevant Slices*", Proceedings of the 30th Annual International Computer Software and Applications Conference, Vol. 1, pp. 411-420, 2006.

[4] Gregg Rothermel, Roland H. Untch, Chengyun Chu, Mary Jean Harrold, "*Prioritizing Test Cases For Regression Testing*", IEEE Transactions on Software Engineering, Vol. 27, No. 10, pp. 929-948, 2001.

[5] Sebastian Elbaum, Alexey G. Malishevsky and Gregg Rothermel, "*Test Case Prioritization: A Family of Empirical Studies*", IEEE Transactions on Software Engineering, Vol. 28, No. 2, pp. 159-182, 2002.

[6] Hyunsook Do., Gregg Rothermel and Alex Kinneer, "*Empirical Studies of Test Case Prioritization in a JUnit Testing Environment*", 15th International Symposium on Software Reliability Engineering(ISSRE), pp.113-124, 2004.

[7] Sebastian Elbaum, Gregg Rothermel, Satya Kanduri and Alexey G. Malishevsky, "*Selecting a Cost-Effective Test Case Prioritization Technique*", Software Quality journal, Vol. 12, No. 3, pp. 185-210, 2004.

[8] Hema Srikanth and laurie Williams, "*Requirements-Based Test Case Prioitization*", Proceeding of the seventh international workshop on Economics-driven software engineering research (EDSER), Vol. 30, No. 4, pp. 1-3, 2005.

[9] Praveen Ranjan Srivastava, "*Test Case Prioritization*", Journal of Theoretical and Applied Information Technology, pp. 178-181, 2008.

[10] Bo Jiang, Zhenyu Zhang, W. K. Chan and T. H. Tse, "*Adaptive Random Test Case Prioritization*", Proceedings of the 24th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 233-244, 2009.

[11] Siripong Roongruangsuwan and Jirapun Daengdej, "*Test Case Prioritization Techniques*", Journal of Theoretical and Applied Information Technology, pp. 45-60, 2010.

[12] Sangeeta Sabharwal, Ritu Sibal and Chayanika Sharma, "*Applying Genetic Algorithm for Prioritization of Test Case Scenarios Derived from UML Diagrams*", IJCSI International Journal of Computer Science Issues, Vol. 8, No. 2, pp. 433-444, 2011.

[13] Sahil Gupta, Himanshi Raperia, Eshan Kapur, Harshpreet Singh and Aseem Kumar, "*A Novel Approach for Test Case Prioritization*" International Journal of Computer Science, Engineering and Applications (IJCSEA) Vol.2, No.3, pp. 53-60, 2012.

[14] Ms. A. Askarunia, Ms. L. Shanmugapriya and Dr. N. Ramaraj, "*Cost and Coverage Metrics for Measuring the Effectiveness of test Case Prioritization Techniques*", ISSN: 0975-3397, 2009.

[15] Rothermel, G., R. H. Untch, C. Chu and M.J. Harrold, "*Test case prioritization: An empirical study*", Proceeding of the 15th International Conference on Software Maintenance, Oxford, England, pp. 179-188, 1999.

[16] Rothermel, G., R. H. Untch, C. Chu and M.J. Harrold, "*Prioritizing test cases for regression testing*" ,IEEE transaction Software Engineering, 27: 929-948, 2001a.

[17] Bryce, R.C. and C. Colbourn, "*Prioritized interaction testing for pair-wise coverage with seeding and constraints*", J. Information Software Tech., 48:960-970, 2006.

[18] Leon, D. and A.Podgurski, "*A comparision of coverage-based and distribution-based techniques for filtering and prioritizing test cases*", Proceeding of the 14th International Symposium on Software Reliability Engineering, IEEE Computer Society, Washington, DC,USA., pp. 442-453, 2003.

[19] Malishevsky,A.,G. Rothernel and S. Elbaum, "*Modelling the cost-benefits tradeoffs for regression testing techniques*", Proceeding of the International conference on software Maintenance, IEEE Computer Society, Washington, DC,USA., pp. 204-204, 2002.

[20] Elbaum, S.,G. Rothermel, S. Kanduri and A.G. Malishevsky, "*Selecting a cost-effective test case prioritization Techniques*", Software Quality Journal, 12: 185-210, 2004.

[21] Malishevsky, A.G., J.R. Ruthruff, G. Rothermel and S. Elbaum, "*Cost-cognizant test case prioritization technical Report*", TR-UNL-CSE-2006-0004, Department of Computer Science and Engineering, University of Nebraska-Lincoln, Lincoln, Nebraska,U.S.A., 2006.

[22] Douglas Hoffman, "*Cost Benefits Analysis of Test Automation*", STARW, Software Quality Methods LLC., 1999.

[23] Kalyana Rao Konda, "*Measuring Defect Removal Accurately*", 2005.

[24] H. Do, G. Rothermel, "*On the use of Mutation faults in Empirical Assessments of Test case prioritization Techniques*", IEEE Trans. Software Eng., Vol. 32, No. 9, 2006.

[25] Zheng Li,Mark Harman and Robert M. Hierons, "*Search Algorithms for Regression Test Case Prioritization*", IEEE Trans. Software Engg., Vol. 33, No. 4, pp. 225-237, 2007.

