International Journal of Trendy Research in Engineering and Technology (IJTRET)
Volume 1 Issue 2 Sep- 2017

A REVIEW ON TEST CASE PRIORITIZATION TECHNIQUES

J. Paul Rajasingh[1], S. Manikandan[2], N. Sankar Ram[3]
Department of Computer Science & Engineering
Sriram Engineering College, India

Abstract:

Test case prioritization aims at finding the ideal ordering of test cases for testing in order to provide test engineers maximum benefit, even if the testing is stopped at some point due to various constraints. Sometimes, the quantum of available testing time is uncertain. For example, market pressures may force the release of a product prior to execution of all the test cases. Test case prioritization can be used either in the initial testing of software or in the regression testing of software. During regression testing, information about previous runs of test cases are normally used to prioritize the test cases for subsequent runs. One of the major goals of test case prioritization is to increase the likelihood of revealing faults earlier in the testing process, especially the most severe faults and hence the rate of fault detection will be improved. The rate of fault detection is a measure of how quickly faults are detected within the testing process. Early detection of faults can provide faster feedbacks and allow the developers to begin addressing faults very soon. In this study, the various test prioritization techniques with their evaluation metrics and the issues raised in each study are presented. This will pave a way to propose a better prioritization technique to minimize the testing cost in distributed applications.

1. Introduction:

Regression testing validates the modified software. It tests whether the changes done to the code for the functionalities and the bugs fixed are negatively impacting the existing functionalities or not. Some of the new features coming in later versions of software may affect the existing and unchanged components of software. Hence, regression testing is crucial to revalidate the existing test cases. Rerunning all of the test cases in a test suite can require a large amount of test effort. An industrial collaborator [1] reported that a test suite with 20000 lines of code required 7 weeks to run. If the human resources are increased, the test duration can be decreased, but still the cost of testing may be expensive. Hence researchers developed various techniques to reduce the cost of regression testing.

The techniques include test suite minimization, test case selection and test case prioritization. Yoo and Harman [4] formally defined and discussed about these techniques. Test suite minimization removes redundant test cases permanently to reduce the size of the test suite. The fault detection capability of the test suite may be decreased due to reduction in the number of test cases. Test case selection techniques select an appropriate subset of the existing test suite based on information about the program, modified version and test suite. It does not remove the test cases, but selects the test cases that are related to the changed portion of the source code. Test case prioritization techniques identify the efficient ordering of test cases to maximize certain goals, such as the rate of fault detection or coverage rate. Though the test suite minimization and test case selection techniques reduce testing time, they can eliminate some significant test cases that can detect certain types of faults and hence leads to increase in the software cost [6]

When the time needed to re-execute an entire test suite is short, test case prioritization may not be cost-effective. When the time needed to re execute an entire test suite is sufficiently long, test case prioritization techniques will be more beneficial. Because test case prioritization techniques use the entire test suite and reduce testing cost by parallelizing debugging and testing activities.

In this paper, the major approaches for prioritizing test cases for regression testing are examined. Also the metrics used in estimating the performance of the approaches are presented.

The next section of this paper precisely describes the test case prioritization problem. Section 3 presents the various performance metrics of test case prioritization. Section 4 presents the various

approaches of test case prioritization. Section 5 presents the overall conclusions and discussions for future work.

2. Test Prioritization:

Test case Prioritization problem [22] was first handled by Wong et al.

Test case prioritization increases the rate of fault detection of test suites, detects the high-risk faults earlier, increases the confidence in the reliability of the system under the test at a faster rate, increases the possibility of regression errors related to specific code changes very early in testing process and increases the coverage of code at a faster rate, allowing code criterion to be met earlier

2.1 Algorithms for test case prioritization

Greedy algorithm, Additional Greedy algorithm, Optimal algorithm, Hill Climbing algorithm, Genetic algorithm, PORT and Ant Colony optimization [4][6][7] are the approaches which have been widely used for test case prioritization.

2.2 Datasets

Public data sets are stored in public repositories like SIR (Software-artifacts Infrastructure Repository). These datasets are publicly available [8]. Private datasets mostly belong to software companies and are not freely distributed as public datasets. Partial datasets are datasets that have been created using data from open source projects and have not been distributed to the community.

3. A Survey on Recent Research in Test Prioritization:

The existing approaches fall into the following categories:

3.1 Coverage based prioritization approach

Coverage based prioritization approach orders test cases based on the coverage of code components. The coverage of code components includes statements and branches. The assumption here is that the maximization of structural coverage will increase the chance of the maximization of fault detection. For example, if a test case A covers more statements or branches over test case B, then test case A may detect more faults than test case B. Statement coverage prioritization prioritizes test cases in terms of the total number of statements covered by them. After counting the number of statements, it sorts the test cases in descending order. Branch coverage prioritization prioritizes test cases in terms of the total number of branches covered by them similar to the previous one.

Coverage based prioritization approach is a white box testing technique that inspects the code directly. On the other hand, black box or functional testing compares the program behavior with requirement specification, without regard to how it works internally.

Greg Rothermel et al [1] [2][3] presented various coverage based prioritization techniques. They used datasets from seven C programs developed by researchers at Siemens Corporation Research for a study of the fault detection capabilities of control-flow and data-flow coverage criteria and one C program developed for the European Space Agency [24]. Average of the percentage of faults detected (APFD) metric was used to evaluate the performance. This metric assumes that the cost of all the test cases and fault are uniform. In practice, test cases and fault costs may vary. Coverage-based white box techniques are applicable for regression testing at the unit levels and are harder to apply on complex systems.

3.2 Cost-aware prioritization approach

Cost-aware prioritization approach incorporates test costs and fault severities into test case prioritization. By enhancing the coverage based techniques developed by [2][3], Elbaum [12] [27] proposed Cost-aware prioritization approach by incorporating the test cost and fault severity of each test case . They used a C program called space which is an interpreter for an array definition language (ADL), developed for the European Space Agency [24]. A new metric average percentage of faults per cost (APFDc) was introduced. This metric takes into account the varying fault severity and test cost.

3.3 History based Approach:

History based approach prioritizes test cases based on the historical factors namely execution history, fault detection effectiveness and coverage of program entities. Based on these factors, priority of the test cases are calculated and the test cases are prioritized.

Jung-Min Kim and Adam Porter[9] proposed an approach that assigns to each test case a selection probability based on test historical execution data. The test cases with higher probabilities are executed till the testing time gets exhausted. Here, the test cases are prioritized according to the test histories namely execution history, fault detection effectiveness and coverage of program entities. Different test histories will yield different test prioritizations. Y. Fazlalizadeh [10][28] combined three kinds of historical information about test cases to form a new

equation. The priority of each test case is calculated and the test cases are scheduled in the decreasing order of the priority values.

These approaches [9][10][28] used the datasets from Siemens and space programs[24] and the Average of the percentage of faults detected (APFD) metric was used to evaluate the performance.

Yu-Chi Huang [23] proposed a history based cost cognizant test case prioritization technique based on the test case costs, fault severities and detected faults of each test case from the latest regression testing. It employs genetic algorithm to find the order with the greatest rate of units of fault severity detected per unit test cost. They considered 2 open source tested programs namely flex and sed obtained from the Software-artifact Infrastructure Repository and the metric average percentage of faults per cost (APFDc) was used to measure the performance.

3.4 Time-aware prioritization approach

Time-aware prioritization approach prioritizes test suite based on the given time constraint. This approach does not prioritize the entire test suite, but it produces a subset of test cases which are prioritized and executed within the given time budget. Walcott [15] presented a time-aware prioritization approach using genetic algorithm. Lu Zhang [31] presented a time-aware prioritization approach using integer linear programming. Two java programs namely JDepend and JTopas were used for experimentation. These programs were available in [8] [30] and the average of the percentage of faults detected (APFD) metric was used to evaluate the performance.

3.5 Probabilistic prioritization approach:

Probabilistic prioritization approach tries to predict the probability that each test case will reveal faults and uses these probabilities to prioritize the test suite. In order to predict this probability, they model regression testing by means of Bayesian network in order to model uncertainty in systems [5]. They used 5 open source java programs namely ant, jmeter, xml, nano and Galileo. These programs and their test suites are all obtained from an infrastructure supporting experimentation[29] and the average of the percentage of faults detected (APFD) metric was used to evaluate the performance.

3.6 Requirement - based prioritization approach:

Requirement-based prioritization approach prioritizes the test cases at system level by considering prioritization factors for each requirement. The factors are assigned values based on the requirement properties. The test cases are then mapped to software requirement and prioritized according to the weight of test cases.

Srikanth[13][15] presented a requirement-based prioritization technique based on the factors namely customer-assigned priority ,requirement changes, fault impact of requirements and implementation complexity. They analyzed 4 student projects to test the effectiveness of their approach.

Krishnamoorthi and Sahaya [25][26] developed a similar approach with additional factors namely completeness, traceability, usability and application flow. Five student projects were system tested to measure the effectiveness of their approach and two industrial projects were chosen to validate the approach.

Both aimed at improving the rate of fault detection. They have taken the total severity of faults detected (TSFD) as the metric.

3.7 Risk based test prioritization approach

Risk-based approaches used in software testing typically focus on risks associated with software requirements [16,17]. Amland [18,19] defined risk exposure as a product of probability of fault occurrence and the cost when the fault occurs in the production. H. Srikanth [20] developed **r**isk based test prioritization approach which prioritizes the test cases at system level. It considers the risks in requirements categories and prioritize requirements categories based on the risk levels of each category. Risk exposure values of the requirements categories are used to prioritize the requirements categories.

Risk exposure (RE) of each requirement category is calculated by multiplying the risk likelihood (RL) of a requirements category and the risk impact (RI) of the requirements category given by

$$RE_i = RL_i * RI_i \quad \text{-------- (1)}$$

RE_i in eq.(1) is the risk exposure of requirements category *i* and RL_i is the risk likelihood of the requirement category *i*, and RI_i is the risk impact of the category *i*. They used the number of test cases of requirements categories to estimate the risk likelihood of requirements categories. In order to estimate the risk impact, they used business criticality or fault proneness. Finally, these risk exposure values of requirements categories are used to prioritize requirements categories.

The approach was applied on an enterprise level IBM analytics application to measure the performance and average of the percentage of faults detected (APFD) metric was used to evaluate the performance. It considers all faults to be equally severe. Also it uses the number of test cases to estimate the probability of fault occurrence as it assumes the number of test cases of a requirement category reflects the functional complexity and the number of functionalities of that requirement category. The presence of higher number of functionalities or more complex functions in a requirement category increases the risk of failures in the requirement category.

	Fault									
Test	1	2	3	4	5	6	7	8	9	10
A	X				X					
B						X	X			
C	X	X	X	X	X	X	X			
D					X					
E								X	X	X

Table 1: Test cases and its detected faults

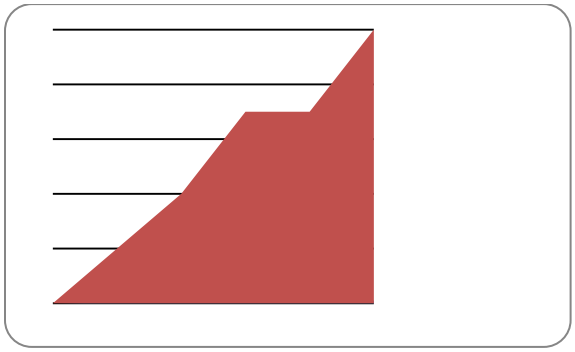

Fig. 1A Test Suite Fraction vs Faults Detected (in %) for Test order ABCDE

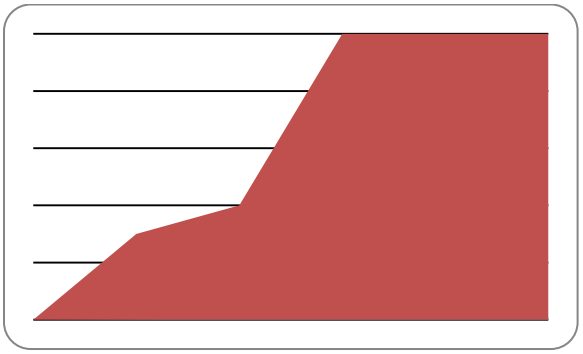

Fig. 1B Test Suite Fraction vs Faults Detected (in %) for Test order EDCBA

4. Evaluation Metric:

4.1. Average Percentage of Faults Detected

The APFD is used to represent the weighted "Average of the percentage of faults Detected" during the execution of the test suite. The APFD values range from 0 to 100; higher values imply faster (better) fault detection rates.

Let us take an example program with 10 faults and a suite of five test cases, A through E, with fault detecting abilities as shown in table1. Suppose the test cases are placed in order ABCDE to form a prioritized test suite A. After running A, 2 of the 10 faults are detected. Thus 20% of the faults have been detected after 20% of A has been used. After running B, two more faults are detected and thus 40% of the faults have been detected after 40% of the test suite have been used. In Figure1A, the area inside the inscribed rectangles represents the weighted percentage of faults detected over the corresponding percentage of the test suite. This area is the prioritized test suite's average percentage faults detected metric (APFD). The APFD is 50%.

When the order of test suite is changed to EDCBA say test suite T_2, the APFD value becomes 64%(Fig.1B) which implies test suite T_2 is better than test suite T_1.

Let T be a test suite containing n sequential test cases, and let F be a set of m faults revealed by T. Let T' be an ordering of T. Let TF_i be the first test case in T' that reveals fault i. The average percentage of faults [1] detected during the execution of test suite is calculated as

$$APFD = 1 - \frac{TF1+TF2+\cdots+TFn}{nm} + \frac{1}{2n}$$

For the test sequence T_1: ABCDE, n=5,m=10

$$APFD = 1 - \frac{(1+3+3+3+1+2+2+5+5+5)}{5*10} + \frac{1}{2(5)}$$

= 50%

For the test sequence T_2: EDCBA

$$APFD = 1 - \frac{(3+3+3+3+2+3+3+1+1+1)}{5*10} + \frac{1}{2(5)}$$

= 64%

4.2.Average Percentage of Faults Detected per Cost (APFDc)

APFD metric assumes that faults have equal severity and test cases have equal costs. Average Percentage of Faults Detected per Cost (APFDc) [12] metric takes into account the varying fault severity and test cost.

Let T be a test suite containing n test cases A,B,…E with costs $t_1,t_2,\ldots t_n$. Let F be a set of faults revealed by T, and let $f_1,f_2,\ldots f_m$ be the severities of those faults. Let TF_i be the first test case in T' that reveals fault i.

The cost cognizant average percentage of faults detected during the execution of T' is given by

$$APFDc = \frac{\sum_{i=1}^{m}(fi\ (\sum_{j=TFi}^{n} tj\ -\ 0.5\ tTFi))}{\sum_{j=1}^{n} ti * \sum_{i=1}^{m} fi}$$

Unlike APFD metric, instead of representing test suite fraction in the horizontal axis, percentage of total test case cost incurred is represented and for percentage of faults detected in the vertical axis, percentage of total fault severity detected is represented.

Under APFD metric, Since all the 10 faults and five test cases are of equal cost, the orders ABCDE and BACDE are equivalent in terms of rate of fault detection. Hence the APFD for these orders is 50%. Suppose B is twice as costly as A demanding 2 minutes to execute whereas A needs one minute. In terms of rate of fault detection, ABCDE is preferable to BACDE.

For the test sequence T_1: ABCDE

APFDc = 28/60

=46.67

For the test sequence T_2: BACDE

APFDc = 26/60

= 43.33

4.3 Average Severity of Faults Detected (ASFD)

A severity value is assigned to each fault. Total severity of faults detected (TSFD) is the sum of severity values of the faults identified. The ASFD for the requirement i (ASFDi) is the ratio of the summation of severity values of faults identified for that requirement to the TSFD[14].

4.4 Total Percentage of Faults Detected (TPFD)

It measures the rate of detection of faults. TPFD is the area under the curve when plotting a graph with the fraction of requirement on X axis and percentage of TSFD on Y axis [14].

4.5 Problem Tracking Reports (PTR) Metric

The PTR metric[12] calculates the percentage of test cases that must be run before all faults have been revealed. PTR is calculated as follows:

$PTR = n_d / n$

Where n is the total number of test cases , n_d is the number of test cases needed to detect all faults in the program.

4.6 Average Percentage Block Coverage (APBC)

The APFD is used to measure the weighted "Average of the percentage of faults detected" during the execution of the test suite. But this is not possible to know the faults exposed by a test case in advance and so this value can not be estimated before testing takes place. Hence Coverage is used as a surrogate measure. APBC[4] measures the rate at which a prioritized test suite covers the blocks.

$$APBC = 1 - \frac{TB1+TB2+\cdots+TB\ n}{nm} + \frac{1}{2n}$$

4.7 Average Percentage Decision Coverage (APDC).

Average percentage decision coverage [4] measures the rate at which a prioritized test suite covers the decisions (branches).

4.8 Average Percentage Statement Coverage (APSC).

Average Percentage Statement Coverage [4] measures the rate at which a prioritized test suite covers the statements.

5. Conclusions and Future Work:

In this paper, we investigated several approaches of test case prioritization. We also presented various metrics and the datasets used to test the software. Through these investigations, the researchers can gain knowledge about the prioritization techniques which can be applied at both black box and white box levels of testing.

Though these test prioritization techniques can improve the rate of fault detection and reduce the cost of regression testing, the majority of them require code coverage information which is very expensive. Utilizing information about the requirements for prioritization can discover more error-prone test cases and reduce the cost. Further, the research can be performed by adding additional factors in the existing approaches, clubbing some of

the approaches or both. These factors will help the researchers to propose a better approach in order to address the current problems in distributed applications.

References:

[1] G. Rothermel, R. H. Untch, Chu Chengyun, and M.J. Harrold, "Prioritizing Test Cases for Regression Testing", IEEE transactions on software engineering, vol.27,No.10, pp.929-948,Oct. 2001.
[2] G. Rothermel, R. H. Untch, Chu Chengyun, and M.J. Harrold, "Test Case Prioritization: An Empirical Study", Proceedings of the International Conference on Software Maintenance, pp. 179, 1999.
[3] Sebastian Elbaum, Alexey G. Malishevsky, G. Rothermel, "Prioritizing Test Cases for Regression Testing", Proceedings of the 2000 ACM SIGSOFT international symposium on Software testing and analysis,vol.25,Issue 5, pp. 102-112, sep. 2000
[4] Z. Li, M. Harman, and R. M. Hierons, "Search Algorithms for Regression Test Case Prioritization", IEEE transactions on software engineering, vol.33, No.4, pp.227-237,April 2007.
[5] S.Mirarab and L.Tahvildari, "An Empirical Study on Bayesian Network-based Approach for Test Case Prioritization", Proc. International conference on software testing, Verification and Validation,2008.
[6] Do H, Tahvildari, and Rothermel," The effects of time constraints on test case prioritization: A series of controlled experiments", IEEE transactions on software Engineering , vol.36,No.5,pp.593-610,Oct 2010
[7] A.Ansari, A.Khan,Alisha khan, K. Mukadam,"Optimised Regression Test using Test Case Prioritization",Proc. Seventh International conference on Communication, Computing and Virtualisation, pp.152-160,2016
[8] Software-artifact Infrastructure Repository, http://sir.unl.edu.
[9] J. M Kim, and A. Porter, "A History-Based Test Prioritization Technique for Regression Testing in Resource Constrained Environments", Proc. 24th International conf. Software Eng., pp. 119-129, May 2002.
[10] A. Khalilian, M. A. Azgomi, Y. Fazlalizadeh, "An improved method for Test Case Prioritization by incorporating historical
test case data", Science of Computer Programming , vol.78,No.1,pp.93-116,Nov. 2012.
[11] H. Park, H. Ryu, and J. Baik, "Historical Value-Based Approach for Cost-cognizant Test Case Prioritization to Improve the Effectiveness of Regression Testing", Proceedings of the international conferences on secure system integration and reliability improvement ,pp.39-46, July 2008.
[12] Alexey G. Malishevsky,Joseph, Gregg Rothermel, Sebastian Elbaum, "Cost-cognizant Test Case Prioritization", Technical Report TR-UNL-CSE-2006-0004
[13] H.Srikanth, L.Williams and J.Osborne," System test case prioritization of new and regression test cases",International symposium of empirical software engineering", 2005
[14] H.Srikanth and L.Williams ," On the economics of requirements-based test case priritization ", Proceedings of the seventh international workshop on Economics-driven software engineering research, 2005
[15]A. Walcott, M. Soffa, G. Kapfhammer, R. Roos, "Time aware test suite prioritization", Proceedings of ACM international symposium on software testing and analysis, pp.17-20, July 2006
[16]J. Bach, "Risk and requirements-based testing ",IEEE computers, pp.120-122,Feb 1999
[17] C. Kaner, J.Bach, "Lessons learned in software testing: A context driven approach",Wiley, New York ,2002
[18] S. Amland," Risk based testing and metrics", International conference on software testing, analysis and review ,Eurostar , November 1999.
[19] S. Amland, "Risk based testing: Risk analysis fundamentals and metrics for software testing including a financial application case study", Journal of systems and software, vol.53,no., pp.287-295, 2000.
[20] H.Srikanth, C. Hettiarachchi, H. Do," Requirements based test prioritization using risk factors: An industrial study", Information and Software Technology, vol.69, pp.71-83, 2016
[21] A.Malishevsky, S.Elbaum and G.Rothermel," Incorporating varying test costs and fault severities into test case prioritization", Proceedings of the 23rd International Conference on Software Engineering, pp. 329-338, 2005
[22] W. Wong, J. Horgan, S. London and H. Agarwal," A study of effective regression testing in practice", Proceedings of eighth international symposium on software reliability engineering, pp. 264-275, 1997
[23] Yu-Chi Huang, Kuan-Li Peng and Chin-YU Huang, " A history-based cost cognizant test case prioritization technique in regression testing", Journal of systems and software , pp.626-637, 2012
[24] M. Hutchins, H. Foster, T. Goradia, and T.Ostrand, "Experiments on the Effectiveness of Dataflow and Control flow-Based Test Adequacy Criteria", Pro. 16th Intl Conf. Software Engg., pp.191-200, May 1994
[25] R. Krishnamoorthi, S. Sahaya Arul Mary," Factor oriented requirement coverage based system test case prioritization of new and regression test cases",Information and Software Technology, vol.51,pp.799-808, 2009.
[26] Krishnamoorthi Ramasamy, S. Sahaya Arul Mary," Incorporating varying requirement priorities and costs in test case prioritization for new and regression testing",International conference on computing, communication and networking, Dec 2008.
[27] Sebastian Elbaum ,Alexey Malishevsky, Gregg Rothermel, "Incorporating Varying Test Costs and Fault Severities into Test Case Prioritization",Proceedings of the 23rd International Conference on Software Engineering, May, 2001.
[28] Y. Fazlalizadeh ,A. Khalilian, M. A. Azgomi, S. Parsa, "Incorporating historical test case performance data and resource constraints into test case prioritization",
Proceedings of the 3rd International conference on tests and

proofs, in: Lecturer notes in computer science, vol.5668, pp.43-576,2009
[29] H. Do, S. Elbaum, and G. Rothermel, "Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact", Empirical Software Engineering: An International Journal, vol.10, pp.405-435,2005
[30] http:// www.clarkware.com / software / JDepend.html
[31] Lu Zhang, Shan-Shan Hou, Chao Guo, Tao Xie, Hong Mei,"Time-aware test case prioritization using Integer Linear Programming", Proceedings of the eighteenth international symposium on Software testing and analysis, pp:213-224, 2009.

Biographies

[1] J Paul Rajasingh completed B.E in Computer Science & Engineering, M.E in Software Engineering and currently doing Ph.D. His areas of interest include software engineering, data mining and data analytics. He is presently working as Assistant Professor in Sriram Engineering college, Tamilnadu, India

[2] S. Manikandan holds M.E in Computer Science and Engineering and Ph.D in Image Processing. He has 15 years of teaching experience in engineering colleges in tamilnadu. He is presently working as Professor & Head in Sriram Engineering college, Tamilnadu, India. His areas of interest include Software Engineering, Image Processing and Operations Research.

[3] N. Sankar Ram holds M.E in Computer Science and Engineering and Ph.D in Software Engineering. He has 18 years of teaching experience in engineering colleges in tamilnadu. He is presently working as Professor in Sriram Engineering college, Tamilnadu, India. His areas of interest include Software Engineering and Network Security