ISSN:2320-0790

An International Journal of Advanced Computer Technology

# A Literature Survey on test case prioritization

**Amit Kumar, Karambir Singh**
Research Scholar, CSE Department, UIET Kurukshetra University
Asst. Prof., CSE Department, UIET Kurukshetra University

**Abstract:** Testing, Analyzing, and Debugging all together are very critical and influential activities for controlling the quality of a software product and it accounts over 50% of the costs associated with the development of whole software systems. Development organizations desire to thoroughly test the software, but exhaustive testing is not possible. Test case prioritization techniques schedule the test cases for execution in an order based on some specific criteria so that the tests with better fault detection capability are executed at an early position. A variety of objective functions are applicable like rate of fault detection, cost involved in testing process, on the basis of users requirement etc. We present the work done in the field of test case prioritization which shows that the latest techniques can be implemented in testing to make the testing more effective and more efficient.

**Keywords**: test case, prioritization, APFD

## INTRODUCTION

Due to technological growth and competitiveness in business software keeps changing and in such environment time to market is a key factor to achieving project success. For a project to be most successful, quality must be maximized while minimizing cost and keeping delivery time short [4]. Quality can be measured by the customer satisfaction with the resulting system based on the requirements that are incorporated successfully in the system [9].To deliver a quality product to the customer each software is tested for which a number of test cases are generated.

Software developers often save the test suites they develop for their software, so that these suites can be reused later as the software evolves. Such use of test suits in the form of regression testing can be seen anywhere in these days in the software industry [1] and, together with other regression testing activities, can account for as much as fifty percent of the cost of software maintenance [2, 3]. Running all test cases in an existing test suite, however, can consume large amount of time and money. For example, one industrial collaborator reports that for one of its products having approximately 20,000 lines of Codes, requires seven weeks to execute the entire test suite. In such cases, Testers may want to order their test cases so that those test cases with the highest priority, according to some criterion, are run first.

Test case prioritization techniques [5] schedule test cases for execution in order according to some criterion. The purpose of this prioritization is to increase the likelihood that they will more closely meet some objective than they would if they were executed in some other order. Test case prioritization can address a wide variety of objectives, including the following:

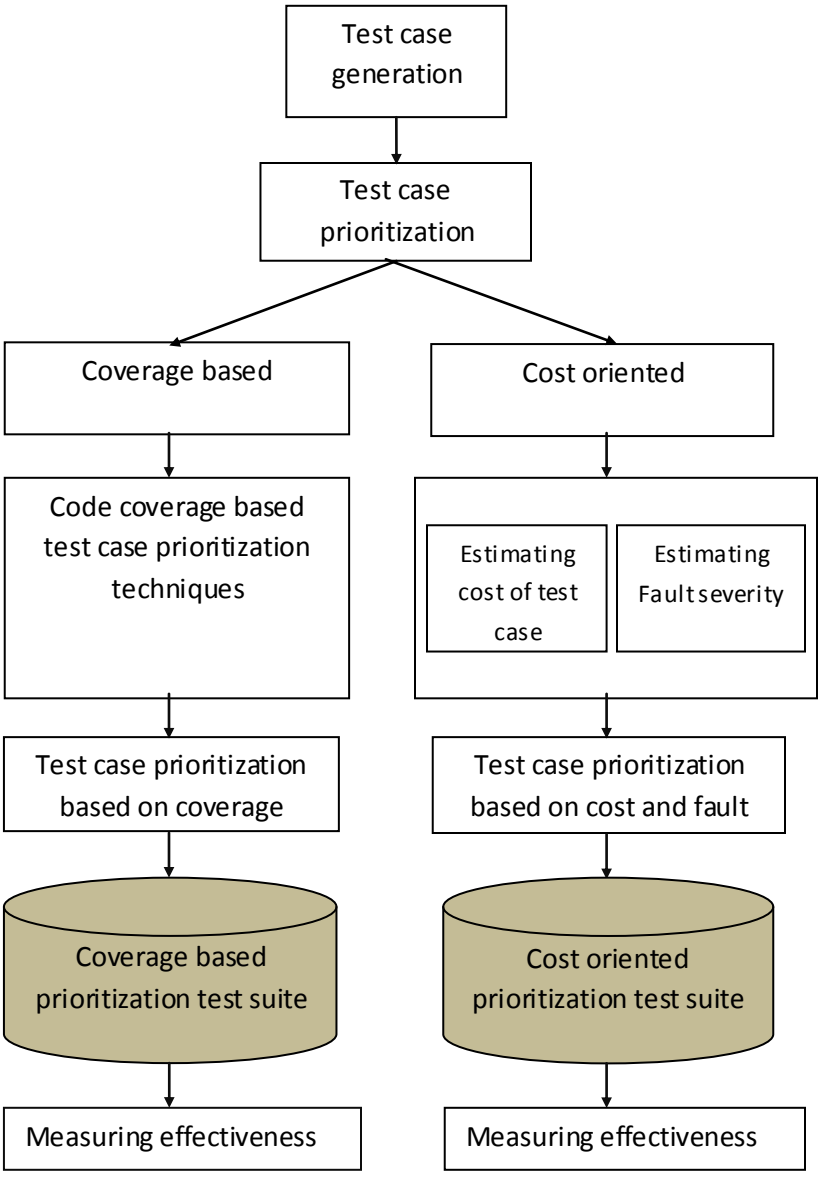
1. Testers may wish to increase the rate of fault detection – that is, the likelihood of revealing faults earlier in a run of regression tests.

2. Testers may wish to increase the rate of detection of high-risk faults based on their severity level, locating those faults earlier in the testing process.

3. Testers may wish to increase the likelihood of revealing regression errors related to specific code changes earlier in the regression testing process.

4. Testers may wish to increase their coverage of coverable code in the system under test at a faster rate.

5. Testers may wish to increase their confidence in the reliability of the system under test at a faster rate.




Test case prioritization problem can be described as:

**Obtained data:** T is a set of test cases, PT is a permutation of the set of all T, f(T) is mapped to the PT in the full array of functions of real numbers.
**Solution**: Find T' ∈ PT,
Such that for all value of T "(T " ∈ PT)(T " != T ') [f (T ') => f (T ")] .

The following diagram shows the general flow followed while testing process.

There are three main test prioritization techniques [8]: 1) control techniques, 2) statement-level techniques, and 3) function-level techniques.

A metric called APFD (Average Percentage of Faults Detected) is used over the life of the test suite to measure, how fast a prioritized test suite detects faults [6]. Its value varies from 0 to 100; higher APFD values means faster fault detection rates [8]. It has 2 assumptions: 1) all faults have equal severity, and 2) all test cases have equal cost. However, in reality faults severity and costs can vary due to

different testing environment and a lot of other factors.

## SURVEY

As mentioned in Section 1, test case prioritization techniques schedule test cases so that those with the highest priority, according to some criterion, are executed earlier in the testing process than lower priority test cases. An advantage of prioritization techniques is that, unlike many other techniques for assisting regression testing, such as regression test selection, they do not discard test cases.
Various prioritization techniques have been proposed.

**Gregg Rothermel** *et al.* [8] conducted an empirical study to prioritize the test cases for various prioritization techniques. The empirical study was conducted with seven C programs with 1000 to 5500 test cases to study the effectiveness of the different test case prioritization techniques. The test cases are considered for entire software program. The effectiveness of the prioritization is measured in terms of average percent of fault detected (APFD). The empirical results show that the test case prioritization techniques improve the rate of fault detection. Rothermel also described the several aspects of the teat case prioritization problem.

a) There are many possible goals of prioritization and to measure the success of a prioritization technique in meeting any such goals, however, one must describe the goal quantitatively and qualitatively.
b) Depending upon the choice of f, the test case prioritization problem may be intractable or non-decidable.
c) Test case prioritization can be used either in the initial testing of software or in the regression testing of software.
d) It is useful to distinguish two varieties of test case prioritization: general test case prioritization and version- specific test case prioritization.
e) It is also possible to integrate test case prioritization with regression test selection or test suite minimization techniques.

In this paper, Gregg Rothermel consider six different test case prioritization techniques which can represent heuristics that can be implemented using software tools; all of these techniques used test coverage information, produced by prior executions of test cases, to prioritize test cases for Subsequent execution. A source of motivation for such approaches is the conjecture that the availability of test execution data can be an asset; however, such approaches also have drawbacks like the assumption that past test execution data can be used to predict, with sufficient accuracy, subsequent execution behavior.

**Kim and Porter** [9] proposed a history- based test case prioritization approach based on the historical fault information. Their experimental results suggested that historical fault information is valuable for improving the effectiveness of the regression testing process in the long term. He also assumed that the test result of each immediately preceding software version has the same importance for the test case prioritization of its successive version across all versions. This leads us to a research question: if the reference value of the test result of the immediately preceding version of the software version is aware for the successive test case prioritization?

**Lionel C. Briand** *et al.* [10] described an empirical investigation in controlled experiment settings, the effectiveness of state-based testing for classes with state-charts. The practical importance of this research was the common use of state-charts to model complex components in object-oriented software. Their results had shown that the mostly used state-based testing technique (roundtrip (RT) path testing) was not sufficient in most situations as major faults remain undetected. This was due to when using a weaker form of roundtrip when guard condition shown several disjunctions and only one of them was exercised. To addresses this types of issues they investigated whether a functional testing techniques could be used in combination with state-based testing in order to achieve a better results for fault detection. They focused on a black-box technique to determine whether category partition (CP) testing that could be used in addition to roundtrip (RT) testing. In this paper compared two different oracle strategies: first strategy was a very specific oracle checking the state of objects, whereas the second strategy was based on the notion of state- invariant. Result had shown the difference between both strategies in forms of cost and fault detection. It should be driven by characteristics of the component that to be tested, for instance criticality, test cost and complexity component.

**Hyunsook Do** *et al.* [11] presented an empirical study of prioritization techniques applied across four different Java programs and provided JUnit test suites to those programs. Although number of different studies on test case prioritization had been conducted

earlier, those studies focused on some specific types of test suites and are done using procedural language, C. while in this paper author applied prioritization techniques to an object-oriented language tested by using the JUnit testing framework, to examine if the results of previous studies generalize to other programming language and testing paradigms as well or not. 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*. [12] in his study suggested 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.

**Zheng Li** *et al*. [13] conducted empirical study to prioritize the test cases using greedy algorithm, additional greedy algorithm, 2 optimal greedy algorithm and genetic algorithm. The main objective of that empirical study was to determine the effectiveness of search algorithm. Various programs were considered ranging from374 to 11,148 lines of code to assess the efficiency of these search algorithms. The empirical results show that the Greedy algorithm performed well in test case prioritization.

**Praveen Ranjan Srivastava** [14] proposed an algorithm for test case prioritization in order to improve regression testing. Analysis was done for the prioritized and non-prioritized cases with the 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*. [15] 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.

**Wong** *et al*. [16] suggested prioritizing test cases according to the criterion of "increasing cost per additional coverage." Although not explicitly stated by the authors, one possible goal of this prioritization is to reveal faults earlier in the testing process. The authors restrict their attention, however, to prioritization of test cases for execution on a specific modified version of a program (what we have termed "version-specific prioritization") and to prioritization of only the subset of test cases selected by a safe regression test selection technique from the test suite for the program. The authors do not specify a mechanism for prioritizing the remaining test cases after full coverage has been achieved. The authors describe a case study in which they applied their technique to a program of over 6,000 lines of executable code (the same program, space, that we use in two of the empirical studies reported in this paper), and evaluated the resulting test suites against 10 faulty versions of that program. They conclude that the technique was cost-effective in that application. He argued that, in addition to historical fault information, the information collected from the source code is also important for test case prioritization. Thus, they suggested prioritizing test cases based on information concerning both historical faults and the source code.

**Zhi Quan Zhou** *et al*. [17] proposed an ART (adaptive random testing) strategy as an improvement of RT (random testing) with an objective of detecting failures early. The basic idea was to more evenly spread test cases over the input domain different

ART algorithms have been proposed to implement the concept of "EVEN*SPREAD.*"The first ART algorithm is known the Fixed Size Candidate Set (FSCS-ART) algorithm. This algorithm maintains a set **E** that stores all test cases that have already been executed, and a set **C** that stores test case candidates initially, a test case is randomly selected from the input domain and run. If no failure is detected, it will be added to **E**. Then a fixed number (normally 10) of test case candidates are randomly sampled from the input domain and put to **C** for each candidate $\mathbf{C}_i$ in **C**, its distance to **E** is measured. To measure the distance from $C_i$ to E, the distance between $C_i$ and each element in E is calculated, and the minimum one among all these distances is taken as the distance from $C_i$ to E. The candidate $C_j$ whose distance to E is the maximum is chosen to be the next test case, and all the other candidates are discarded. Intuitively, test cases thus selected are far apart from each other. This process is repeated until the testing stopping criterion is met. The time complexity of this algorithm is in $O(n^2)$ where n is the number of test cases executed. It has been demonstrated that FSCS-ART can achieve a much lower *F-measure* than RT, where F measure is the number of test cases executed to detect the first failure.

**Chu-Ti Lin** *et al. [18]* this paper describes the techniques for the prioritization of test cases on the basis of their history. Most of the existing test case prioritization approaches are code-based, in which the testing of each software version is considered as an independent process and most of them are memory-less in that they model regression testing as a one- time activity rather than a continuous process. But Actually, the test results of the preceding software versions may be useful for scheduling the test cases of the later software versions. Some researchers have proposed history-based approaches to address this issue, but they assumed that the immediately preceding test result provides the same reference value for prioritizing the test cases of the successive software version across the entire lifetime of the software development process. The experimental results indicate that, in comparison to existing approaches, the presented one can schedule test cases more effectively. Chu-Ti Lin collected the statistics about the Siemens programs from the Software-artifact Infrastructure Repository (SIR), that are frequently used to compare different test case prioritization methods. After analyzing the test results of all versions of the Siemens programs, he found that, for the test cases detecting faults in a specific version, there is a higher probability that they will detect faults again in the successive version. This confirms that the historical fault information deserves to be considered when prioritizing the tests during future regression testing

**Paolo tonella** *et al.* [19] In this paper Tonella described that test case prioritization has number of different objectives and among them all objectives the most important one is probably maximizing the rate of fault detection, which consists of revealing faults as early as possible in the testing process, it can be calculated by using the APFD. APFD is acronym for average percentage of fault detection. Higher the value of APFD means earlier the faults are detected. APFD can be computed according to the following equation:

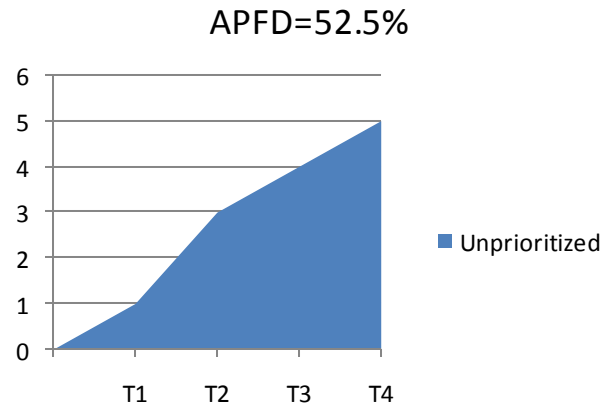
$$APFD = 1 - \frac{\sum_{i=1}^{m} TFi}{nm} + \frac{1}{2n}$$

Where, *m* is the the number of exposed faults

*n* is the total number of test cases and

*TFi* is the position of first test case in T that exposes fault i.

| test | Faults detected | | | | |
|---|---|---|---|---|---|
| T1 | | | | | X |
| T2 | | | X | X | |
| T3 | X | | X | | X |
| T4 | X | X | X | X | |



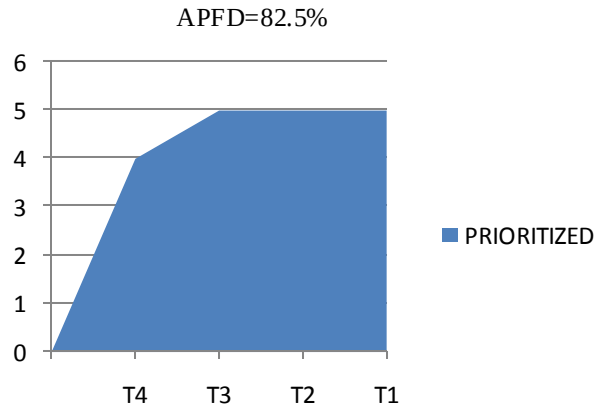



**Figure 1.** APFD is higher for the order that reveal the faults earlier.

As shown in Figure 1, the APFD tends to have high values when the ordering of the test cases is such as to reveal most faults at early stage. The APFD is the portion of area below the curve in Figure 1.

He proposed a test case prioritization technique that takes advantage of user knowledge through a machine learning algorithm, Case-Based Ranking (CBR). CBR elicits just relative priority information from the user, in the form of pair-wise test case comparisons+. User input is integrated with multiple prioritization indexes, in an iterative process that successively filters the test case ordering. Preliminary results on a case study indicate that CBR overcomes previous approaches and, for moderate suite size, gets very close to the optimal solution. According to **Paolo tonella** CBR learns the target ranking from two inputs: (1) a set of possibly partial indicators of priority and (2) pair-wise comparisons elicited from the user (cases). On one hand, all the information that can be gathered automatically about the test cases (coverage levels, fault proneness metrics, etc.) is used by CBR to approximate the target ranking. On the other hand, the user is involved in the prioritization process to resolve the cases where contradictory or in-sufficient data are available. The contribution required from the user consists of very local information and has the form of a pair-wise comparison. In given number of test cases, the user is requested to indicate the one that should be given higher priority. No quantification and no global evaluation is required. No consistency, such as transitivity, in the elicitation process is assumed. CBR operates iteratively and it produces a provisional ordering at each iteration. Thus, prioritization can be stopped at any time and CBR provides the user with the last ordering produced. Thus, the human effort dedicated to the prioritization process can be calibrated arbitrarily.

## FUTURE WORK

A lot of work has been done in this field but still we have a long way to go to achieve a target of 90 percent or beyond that accuracy level. In future we can work on generating some new techniques which can help the developer to detect the error much before it is introduced so that it can be removed in the early phases of development which costs very low.

In previous empirical studies of test case prioritization the researchers only concentrated on one objective i.e. average rate of fault detection. In order to carry out some general results, other objectives for prioritization also need to be considered. Truly generic results can be achieved through additional understanding and careful control on various factors (e.g., number of faults, testing time, number of test cases, subject program etc.) these factors affect the cost-effectiveness of reduction and prioritization techniques. We can also use the clustering of test cases to improve the fault detection rate of our test suits.

## REFERENCES

[1] K. Onoma, W.-T.Tsai, M. Poonawala, and H. Suganuma.Regression testing in an industrial environment.*Comm. Of the ACM*, 41(5):81–86, May 1988.

[2] H. Leung and L.White. Insights Into Regression Testing. In *Proc. of the Conf. on Softw.Maint.*, pages 60–69, Oct. 1989.

[3] B. Beizer. *Softw.Testing Techniques*. Van Nostrand Reinhold, New York, NY, 1990.

[4] J. Karlsson and K. Ryan, "A Cost-Value Approach for Prioritizing Requirements," *IEEE Software*, vol. 14, no. 5, pp. 67-74, Sep-Oct 1997.

[5] W.Wong, J. Horgan, S. London, and H. Agrawal. A study of effective regression testing in practice. In *Proc. of the Eighth Intl. Symp. onSoftw. Rel. Engr.*, pages 230–238, Nov. 1997.

[6] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold, "Test case prioritization: an empirical study," In Proceedings of the IEEE International Conference on Software Maintenance (ICSM'99), Washington DC USA, pp. 179–189, 1999.

[7] G. Rothermel, Mary Jean Harrold, and JeinayDedhia,

"Regression Test selection for C++ Software", Research
Article Software Testing, Verification and Reliability,"
John Wiley & Sons, Vol. 10, No. 2, pp 77 – 109, Jun 2000.

[8] G. Rothermel, R. Untch, C. Chu, and M.J. Harrold.
"Prioritizing Test Cases for Regression Testing," *IEEE*
*Trans. Software Eng.*, Vol. 27, No. 10, pp 929-948, Oct. 2001.

[9] J. M. Kim and A. Porter, "A history-based test prioritization technique for regression testing in resource constrained environments," Proc. ACM/IEEE Conf. Software Engineering, May 2002, pp. 119-129.

[10] Lionel C. Briand, Massimiliano Di Penta and Yvan Labiche, "Assessing and Improving State-Based Class Testing: A Series of Experiments", IEEE Transactions on Software Engineering, Vol. 30, No.1, pp. 770-783, 2004.

[11] 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.

[12] 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.

[13] Zheng Li, Mark Harman, and Robert M. Hierons, "Search Algorithm for Regression Test Case Prioritization, *IEEE Trans. Software Eng.*, Vol. 33, No. 4, pp 25-37, 2007.

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

[15] 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.

[16] W. N. Liu, C. Y. Huang, C. T. Lin, and P. S. Wang, "An evaluation of applying testing coverage information to historical-value-based approach for test case prioritization," Proc. Asia-Pacific Symp. Internetware, December 2011, pp. 73-81.

[17] Zhi Quan Zhou†, Arnaldo Sinaga‡and Willy Susilo "On the Fault-Detection Capabilities of Adaptive Random Test Case Prioritization:
Case Studies with Large Test Suites"IEEE Software Engg,2012

[18] Chu-Ti Lin, Cheng-Ding Chen, Chang-Shi Tsai1, Gregory M. Kapfhammer "History-based Test Case Prioritization with Software Version Awareness" IEEE conference publishing services, 2013

[19] Paolo Tonella, Paolo Avesani, Angelo Susi, "Using the Case-Based Ranking Methodology for Test Case Prioritization" 22nd IEEE International Conference on Software Maintenance (ICSM'06)