DOI: 10.7763/IPEDR. 2014. V75. 5

# Systematic Review on Quality Assessment of Test Case Prioritization Techniques in Software Testing

Akash Goyal[1+], Aakanksha Sharaff[2], Dr. N. K. Nagwani[3]

B.Tech. Student, Assistant Professor, Assistant Professor

Department of Computer Science & Engineering NIT Raipur (C.G.)

**Abstract.** Activities performed on a software product after its release for use is called Software Maintenance. These activities or actions deal with the changes that are often vital during this stage of the software life cycle. Software testing, as a software quality assurance, is becoming more and more important. In the software life cycle, regression testing need to be done each time the code has changed. Most of the test case prioritization techniques that we have studied can be categorized into either of customer requirement based, coverage based, cost effective based or chronographic history based methods. Also there are techniques which falls into categories different from the one mentioned above. The test case prioritization techniques can be compared on various grounds like on the basis of use of factors, or on the basis of degree of coverage like statement coverage, branch coverage, loop coverage, or on the basis of fault exposing potential, etc. This paper discusses some selected test case prioritization techniques for regression testing presented by researchers.

**Keywords:** Software Maintenance, Regression Testing, Test Case Prioritization

## 1. Introduction

Regression testing is the procedure of retesting the modified portions of the software and assuring that no new bugs are generated into the previously tested code. The test case library (test suite) becomes very large. Thus we need to prioritize the test cases, so as to run some selected test cases which can provide the expected result similar to complete test suite, and can detect faults earlier. Test case prioritization orders tests for execution so that the test cases with the higher priority, based on some criterion, are executed before lower priority test cases. Elbaum [2] formally defined Test Case Prioritization as:-

*Given: T, a test suite; PT, the set of permutations of T; f, a function from PT to the real numbers.*

**Problem:** *Find T' belongs to PT such that (for all T") (T" belongs to PT) (T" ≠ T') [f (T') ≥ f (T")]*

The earlier techniques presented by the researchers mainly dealt with code coverage. Other researchers presented techniques based on several factors like cost factors, time factors, requirement factors, etc. to prioritize test cases. We will discuss the various Code Coverage Techniques, factor based techniques, and some other techniques.

The rest of the paper is organized as follows: Section II describes the *Related Work*. Section III describes *Discussion and Results* containing the advantages and disadvantages of various prioritization techniques. Section IV presents C*onclusion and Future Work*.

## 2. Related Work

### 2.1. Code Coverage Based Techniques

Rothermel [1] has described several techniques for prioritizing test cases and find the empirical results, measuring the effectiveness of the techniques for improving rate of fault detection. These techniques include:-

i. **No prioritization**: **-** This let the system consider "untreated" test suites.

[+] Corresponding author.
*E-mail address*: *akashgoyal7@gmail.com*

ii. **Random prioritization**: - Test Cases in a test suite are ordered randomly.

iii. **Optimal prioritization**: - Determine an optimal ordering of test cases in a test suite for maximizing that suite's rate of fault detection.

iv. **Total branch coverage prioritization**: - Prioritize the test cases according to the total number of branches they cover.

v. **Additional branch coverage prioritization**: - Prioritize the test cases according to the number of branches not yet covered.

vi. **Total fault-exposing-potential (FEP) prioritization**: - Prioritize tests in order of total probability of exposing faults.

vii. **Additional fault-exposing-potential (FEP) prioritization**: - In additional FEP prioritization, after selecting a test case t, we lower the award values for all other test cases that exercise statements exercised by t.

viii. **Total statement coverage prioritization**: - Prioritize the test cases according to the total number of statements they cover.

ix. **Additional statement coverage prioritization**: - Prioritize the test cases according to the number of statements not yet covered.

Elbaum [2] has explained 18 different test case prioritization techniques and classified them into three groups: *Comparator Group*, *Statement level Group*, and *Function level Group*. The composition of each group is as follows.

The comparator group contains:

i. **Random Ordering:** ordering on random basis.

ii. **Optimal Ordering:** ordering optimizes the rate of fault detection.

The statement level group contains:

iii. **Total statement coverage:** ordering on basis of statements coverage.

iv. **Additional statement coverage:** ordering on basis of statements not yet covered.

v. **Total FEP:** ordering on the basis of probability of exposing faults.

vi. **Additional FEP:** similar to Total FEP but considers previous test cases.

The function level group contains:

vii. **Total function coverage:** ordering on basis of functions  coverage.

viii. **Additional function coverage:** ordering on basis of functions not yet covered.

ix. **Total FEP (function level):** ordering on the basis of probability of exposing faults.

x. **Additional FEP (function level):** similar to Total FEP but considers previous test cases.

xi. **Total fault index (FI):** ordering on the basis of probability of fault existence.

xii. **Additional fault-index (FI):** similar to Total FI but considers previous test cases.

xiii. **Total FI with FEP coverage:** ordering on basis of combined probabilities of fault existence and fault exposure.

xiv. **Additional FI with FEP coverage:** similar to Total FI with FEP coverage but considers previous test cases.

xv. **Total DIFF:** prioritization on the basis of probability of fault existence.

xvi. **Additional DIFF:** similar to Total DIFF but considers previous test cases.

xvii. **Total DIFF with FEP:** ordering on basis of combined probabilities of fault existence and fault exposure.

xviii. **Additional DIFF with FEP:** similar to Total DIFF with FEP coverage but considers previous test cases.

The *Total* techniques prioritize test cases based on the information available at the start of prioritization, whereas the *Additional* techniques adjust their efforts to consider the effects of previous test cases.

Gantait [4] presented a new approach using UML Activity Diagrams to generate and prioritize test cases. He defined two algorithms: one to generate the set of test cases form Activity Diagram, other to assign priority to those test cases. The test suite includes various flows of the activity diagram. The prioritization technique uses algorithms which assign weights to all the flows (test cases) and then find the minimal subset of the generated Test Suite which will cover the complete Activity Graph.

## 2.2. Factor based techniques

Shrikanth [5] presented requirements-based test case prioritization approach, with one of the preliminary research objective to create the minimal set of PORT (Prioritization of Requirements for Testing) prioritization factors that can be used effectively for test case prioritization.. They proposed to use several factors to rank the test cases. Those factors are: - (1) customer-assigned priority (CP) based on requirements, (2) Requirements complexity (RC), (3) Requirements volatility (RV). They assigned value (1 to 10) to each factor for the measurement. They stated that higher factor values indicate a need for prioritization of test case related to that requirement. Weight prioritization (WP) measures the importance of testing a requirement earlier, and it is represented by the equation (1): -

$$WP = \sum_{PF=1}^{n} \left( PF_{value} * PF_{weight} \right) \qquad (1)$$

Where *n* is the total number of test cases, WP denotes weight prioritization, PFvalue is the value of each factor like CP, RC and RV, and PFweight is the weight of each factor like CP, RC and RV. Test cases are ordered such that the test cases with high WP are executed before others.

Siripong [6] introduced 4C classification for the existing prioritization techniques. He classified techniques into Customer requirement based, Coverage based, Cost based and Chronographic history based. The author also described two methods: *MTSSP* (Multiple Test Suite Same Priority) & *MTSPM* (Multiple Test Suite Prioritization Method). The first is to prioritize multiple test suites and test cases with same priority, and the second to effectively prioritize multiple test suites. These methods are based on a set of proposed Cost factors, Time factors, Defect factors and Complex factors.

**MTSSP Method**: - This method aims to solve the problem of multiple test suites assigned the same priority. The test suite is fed in, and then the test cases in the suite are prioritized on the basis of the Defect factors. If the problem is not solved yet, then the suite is prioritized on the basis of the Time factors. Again if the problem is not solved, then the suite is prioritized on the basis of the Cost factors. If this attempt fails then we go with the Complex factors. If the problem is not solved by Complex Factors, then the suite is prioritized by a random method.

**MTSPM Method**: - This method aims at effectively prioritizing multiple test suites. The multiple test suites are fed in, and then they are prioritized using only the Time & Cost Factors. If the test suites still have the same priority then random ordering is done. Otherwise, MTSSP method is used to prioritize the test cases in the suite. If the cases still have same priority after the MTSSP method, then prioritize them using MTSPM method.

## 2.3. Historical data based Techniques

Kim [7] proposed a new technique that bases prioritization on historical execution data. The prior performance information of test cases is used to find the likelihood of using that test case in the current testing session. The authors approach is based on the ideas from statistical quality control and statistical forecasting.

The author described the selection probabilities of each test case *tc* at time *t* as $P_{tc},_{t}(H_{tc}, \alpha)$, where $H_{tc}$ is a set of *t* time-ordered observations $\{h_1, h_2, \ldots\ldots\ldots, h_t\}$ drawn from previous runs of *tc*, and $\alpha$ is a smoothing constant used to weight individual history observations. The higher values of $\alpha$ emphasize recent observations; while lower values of $\alpha$ emphasize older observations. These values are used as probabilities after normalization. Equations (2) and (3) represent the general form of P as: -

$$P_0 = h_1 \qquad (2)$$

$$P_k = \alpha h_k + (1 - \alpha)P_{k-1}, 0 \le \alpha \le 1,\ k \ge 1 \quad (3)$$

## 2.4. Other Techniques

Srivastava [3] suggested prioritizing test cases according to the criterion of increased APFD (Average percentage of Faults detected) value. He proposed a new technique which could be able to calculate the average number of faults found per minute by a test case and using this value sorts the test cases in decreasing order. APFD is calculated by the equation (4) represented as:

$$APFD = 1 - \left(\frac{(TF_1 + TF_2 + \cdots\cdots + TF_m)}{nm}\right) + \left(\frac{1}{2n}\right) \quad (4)$$

Where *T* is the test suite under evaluation, *m* is the number of faults contained in the program under test, *n* is the total number of test cases, and $TF_i$ is the position of the first test in *T* that exposes fault *i*.

## 3. Discussion and Results

The results of Rothermel [1] techniques vary across the individual programs. On the basis of experiment given in paper, it was shown that additional FEP prioritization outperformed all prioritization techniques based on coverage, and total FEP prioritization outperformed all coverage-based techniques other than total branch coverage prioritization. Elbaum [2] has extended the Rothermel [1] classification. He put the different 18 techniques into three different groups according to granularity. This makes it easy to specify a technique for a particular problem. Gantait [4] method generated test tool specific test cases from the activity models created early in the development cycle, and also it reduces the number of test cases to a considerable extent while covering all transitions in the activity model. The author mentioned that his method has some minus points like - all the selected paths may not be feasible, a detailed analysis of states is required to understand the possible paths, it is based on possibility of usage of a path at runtime, and a separate flow weight assignment approach is needed.

Shrikanth [5] method could improve the effectiveness of testing activities, as it reduces the effort utilized for Test Case Prioritization in comparison to coverage-based techniques, and focuses on functionalities that are of highest value to the customer, and improves the rate of detection of severe faults. Methods of Siripong [6] can prioritize a set of test cases in case there are multiple cases with the same priority weight values, and, can prioritize multiple test suites. These techniques make use of 13 different factors which are grouped into cost, time, defect and complex factors. Thus, these techniques are dependent on factors and require the values of various factors before processing. Kim [7] presented a technique with memory element, which was different from the other memory less techniques. This technique prioritises test cases using the historical test execution data. Srivastava [3] method requires prior knowledge of faults. Calculation of APFD is possible with prior knowledge of fault. Therefore, APFD calculations are used only for evaluation.

## 4. Conclusion and Future Work

Various test case prioritization techniques are discussed in this paper. The various techniques of test case prioritisation are grouped into either of code coverage techniques, factor based techniques, historical data based techniques and other techniques to make it easy to understand the primary requirement & working scope of these techniques. This paper also discusses the plus and minus points of various techniques, on the basis of experiments done by the researchers who have proposed these techniques.

In factor-based techniques, work can be done to find answer for the question: What are the different ways of assigning values to various factors? When assigning values to factors, it can be done that the values are assigned on the basis of a survey-poll of many developers, rather than relying on a single developer. This is because their opinion may not be the same about assigning the priority to the test cases. Some methods which are less complex and easier to implement are needed. Work can be done in the field of historical-based-techniques to get good methods. This is because with the use of historical data, prediction becomes more accurate.

## 5. References

[1] Gregg Rothermel, Roland H. Untch, Chengyun Chu, Mary Jean Harrold. Test Case Prioritization: An Empirical Study. Proceedings of International Conference on Software Maintenance, 1999.

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

[3] Praveen Ranjan Srivastava. Test Case Prioritization. JATIT, 2008.

[4] Amritanjan Gantait. Test case Generation and Prioritization from UML Models. IEEE, 2011.

[5] Hema Shrikanth, Laurie Williams. Requirements based Test Case Prioritization. ACM SIGSOFT Software Engineering, 2005, pages 1-3.

[6] Siripong Roongruangsuwan and Jirapun Daengdej. Test Case Prioritization Techniques. JATIT, 2005-2010.

[7] Jung-Min Kim, Adam Porter. A history based test prioritization technique for regression testing in resource constrained environments. ICSE, 2002.

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

[9] The Unified Modeling Language User Guide, Second Edition. Grady Booch, James Rumbaugh, Ivar Jacobson. 2005.