



A Test Case Prioritization Method with Weight Factors in Regression Testing Based on Measurement Metrics

Thillaikarasi Muthusamy¹

¹(Assistant Professor)

Department of Computer Science and Engg,
Faculty of Engineering and Technology
Annamalai University, Annamalai Nagar,
Tamilnadu, India-608002

Dr. Seetharaman.K²

²(Associate Professor)

Department of Computer Science and Engg,
Faculty of Engineering and Technology
Annamalai University, Annamalai Nagar,
Tamilnadu, India-608002

Abstract--- *Scheduling test cases by using test case prioritization technique enhances their efficiency of attaining some performance criteria. The rate at which the errors are detected within the testing process is one such criteria. An enhanced rate of fault detection during testing can provide quicker feedback on the system under test thereby allowing s/w engineers to rectify errors before usual time. An application of prioritization techniques includes regression testing of s/w which has undergone alterations. To optimize regression testing s/w testers may assign test case preferences so that to some extent the more significant ones are run earlier in the regression testing process. Software testing and retesting is an inherent part of software development as far as the errors are detected at the earliest that may not change the sequence hence gaining confidence. Regression testing has been proved to be crucial stage of software testing. Regression test prioritization techniques manipulates the execution of test case so that faults are detected at the earliest. To achieve performance requirements test cases with higher priority are executed than those with lower priority by test case prioritization techniques. This proposed test case prioritization algorithm prioritizes the test cases based on four groups of practical weight factors such as Time factor, Defect factors, Requirement factor and complexity factors. The proposed technique is validated with three different validation metrics and is experimented using two projects. The algorithm illustrated detects serious errors at earlier phases of testing process and effectiveness between prioritized and un prioritized test cases is compared using ASFD.*

KEY WORDS: *Regression Testing, Test case, Test case prioritization, Fault severity, Rate of fault detection.*

I. INTRODUCTION

In the software development life cycle, regression testing is a necessary component that seeks to uncover software errors after changes to the program. A set of test cases should be written adequately to test the software. When the software is altered, a new set of test cases are added to the test suite to test the changed requirements. It is impractical and inefficient to re-execute every test for every program functions if a change occurs. This tends to increase the size of test suite, cost and time constraints. The problem of regression test case selection is solved by prioritizing test cases. Test case prioritization techniques can organizes the test cases in an order to increases the effectiveness of testing. It also avoids the test suite minimization problems. Recently, the experimental studies of Dennis Jeffrey and Neelam Gupta shows that fault detection capability can be enhanced by selective retaining of test cases. Several techniques for prioritizing test cases have been proposed that are described in the next section of related work. This paper, investigates the use of an evolutionary approach for prioritizing the test cases in regression testing. The main objective is to run test cases based on practical weight factors to increases the possibility of fault detection and detects the severe faults in early stages of testing life cycle. The rest of this paper is structured as follows: Section 2 presents a concise review of related works. Section 3 briefly presents the prioritization technique for prioritizing test cases based on weight factors. Section 4 reports the evaluation of proposed technique based on three metrics. Section 5 reports the experimental results for two set of projects. The final section presents a conclusion with a discussion on further work.

II. RELATED WORK

Test case prioritization [1] is an important kind of regression testing technique [1] [2]. Test case prioritization approaches typically sort existing test cases for regression testing according to attain performance goals. The metric of Average Percentage Of Fault Detected (APFD) is widely used for evaluating test case prioritization techniques. Researchers have used various prioritization techniques to measure APFD values and found it produces statistically significant results. The APFD is a measure that the average number of faults identified in a given test suite. The APFD values ranges from 0 to 100 and the area under the curve by plotting percentage of fault detected against the percentage of test cases executed.

The prioritization techniques presented by Rothermal and Elbaum [1] involves coverage of test cases. Ranking of test cases are given based on statement/function/branches covered by each test cases. Rothermal et.al [1] used TCP

techniques in 3 different groups such that comparator group, statement level group, functional level group. Each containing fine granularity techniques such as random ordering, optimal ordering, total statement (total-st), total branch (total-br), total function (total-fn), additional statement (addtl-st), additional branch (addtl-br), additional function (addtl-fn). Total statement (total-st) TCP techniques measure the coverage of statements in a program. Prioritization of test cases is done by the total number of statements they cover and sorted in the order of coverage achieved. Total branch and total function similar to total statement (total-st) except that they use function and branch coverage instead of statement coverage information. Additional statement (addtl-st) TCP techniques select a test case that covers the maximum number of statements not covered in each round. When two test cases cover the same number of additional statements in a round, it randomly picks one, the additional function (addtl-fn) and additional branch (addtl-br) are similar to additional statement (addtl-st) except that they are using branch coverage and function coverage information respectively.

The code coverage strategies [1] were measured using weighted average percentage of fault detected (APFD) [2], average percentage of branch covered (APBC), average percentage of decision covered (APDC) and average percentage of statement covered (APSC). APFD is the measure of rate of fault detected by a test suite. The potential goal of the above technique is to increase the rate of fault detection at earlier stages of testing process [2].

Kim and Porter [4] introduced a technique that uses historical execution of data for test case prioritization in a regression testing. In this technique, Test case prioritization problem is considered as a probabilistic approach so that selection probability of each test case in a test suite is based on the execution history that will be used in the next test session. Test cases are prioritized according to the execution history, faulty detection effectiveness and the coverage of the program respectively.

Yu-chi Huang et al [7] proposed a cost-cognizant prioritization technique according to various test costs and fault severities of each test cases and proposes genetic algorithm to determine the effective prioritization order. It is validated using two UNIX programs and evaluate the effectiveness using APFD.

Qu et al [6] proposed an algorithm based on test history and run-time information to prioritize test cases in run-time environment. This algorithm categorizes test cases by the faults exposed and then prioritizes test cases according to the results from the related test cases. Park et al [5] proposed a historical value model which quantified the historical value of test cases based on empirical test case costs and the historical cost-cognizant test case prioritization.

Harrold et al [8] approaches the static program slicing to detect definition-use associations that are affected by program changes. Rothermal and Harrold [9] presents a model for changed code. In this approach, constructs control flow graph for a procedure and its modified version to select test cases that executes changed code. Agrawal et al [11] proposed dynamic slice based and the relevant slice based approaches to determine the test cases in a test suite on which the new and old program produces different outputs.

Dennis Jefferey and Neelam Gupta [10] present a new approach to test case prioritization using relevant slices. Implementation of this technique is done by three different heuristics based on relevant slicing and compares the effectiveness with traditional techniques only for requirement coverage.

The topic of test suite minimization [13] is closely related to the test case prioritization. A test suite of test cases where a set of requirements must be satisfied the desired test coverage of the program and subsets of the test suite where each subset is associated redundant test cases to be removed so that a minimal test suite can be constructed.

Other approaches includes modeling the cost-benefits for regression testing (Malishevsky et al., [12]) measuring the impact of test case reduction on fault detection capability and fault detection capability with the branch coverage techniques. These techniques permanently discard a subset of test cases that will need for regression testing while test case prioritization technique preserves all test cases and re-orders them in test list.

III. PROPOSED PRIORITIZATION TECHNIQUE

In this section we present the proposed set of practical prioritization factors and the prioritization algorithm

A. Factors To Be Considered For Prioritization

We consider the comprehensive set of weight factors for test case prioritization so that the software test engineers should not ignore these practical weight factors while running a test case prioritization process. We proposes 10 factors classified into 4 groups which are (1) Time factors, (2) Defect factors, (3) Requirement Factors, (4) Complexity Factor.

1) Time factors

In our proposed prioritization technique, we consider two time factors are as follows:

- **Execution Time (ET).**

It is the subjective measure of which total time required for the execution of test suite. Weightage can be assigned by three states such as high, medium, low ranges from 1 to 10 scale assignment. High denotes the scale from 8 to 10 points, medium denotes the scale from 4 to 7 points and low denotes the scale from 1 to 3 points. Douglas suggested this time consumption factor is a common factor for software testing and applied for test case prioritization.

- **Validation Time (VT).**

It is the measure of which the total time required for validating the expected result and actual result. Weightage can be ranged from 1 to 10 scale assignment. Kalyana (2005) described that the time consumption for validation factor is the most important metrics to validate results and find a defect.

2) Defect Factors

In test case prioritization test cases detects bugs should have higher priority, so that those bugs will be fixed and required to re-test again. This factor can be referred as defect discovery rate most widely used metrics in software testing Rajib (2006)

- *Defect Occurrence (DO).*

It is a measurement of how many test cases will detect defects after its execution. Scale assignment is done by true and false conditions where true denotes 10 points and false denotes 0 points. Julie and mark reported that this factor is widely used in defect measurement system and is recorded in defect reports(Offutt et al.,1995).

- *Defect Impact (DI).*

It is a dimension for classifying seriousness for defects. When the system evolves to several versions the development team can use the empirical data collected from previous version to identify the specific requirement that can likely to be error prone. In this work we propose to calculate defect impact based on the severity of the fault identified in the previous run. When t number of faults identified by the ith test case then Severity value of ith test case can be shown below,

$$S_i = \sum_{j=1}^t SV \quad \dots (2)$$

Max(S) is the severity value of test case among all the test cases then the Defect Impact of ith test case can be calculated as,

$$DI_i = \left(\frac{S_i}{Max(S)} \right) * 10 \quad \dots (3)$$

The severity level of a defect indicates the business impact factor for the end user. The high severity defect means low product/software quality and vice versa.

3) *Requirement Factors*

In our proposed technique, we consider four requirement factors for the test cases as they much impact on new software. Each of the four factors will now be discussed.

- *Customer Assigned priority (CP).*

It is a measure of the importance of customer requirement. So, the requirement with highest customer importance should be tested early to improve customer satisfaction. The customer assigns the values for each requirement ranging from 1 to 10 where 10 denote highest customer priority.

- *Implementation complexity (IC).*

It is a measure of the complexity in implementation of requirement by development team. Requirements with high implementation complexity will have high number of faults. Each Requirement is assigned to the values ranging from 1 to 10 where 10 indicate high implementation complexity.

- *Requirement Change (RC).*

It is based on total number of times a requirement has been changed in their development cycle. 50% of faults in the projects is identified in requirement phases due to changing in requirement stages. The volatility changes for all the requirements are normalized to 10 point scale.

- *Requirement Coverage (ReqCov).*

It is the subjective measure of total number of requirements covered by each test case in a test suite. We can assign weight in 10 point scale where 1 is the minimum value and 10 is the maximum value. Literature review (Lormans and van Deursen,2005) shows that requirement coverage can helps to validate that all requirements are implemented in the system.

4) *Complexity Factor:*

The complexity factor determines the total effort required to execute the test cases. Several studies (Tsui et al.,2006; Tai,1980) shows that the complexity of test cases is one of the most important factors for regression test case prioritization. We can consider the following complexity factors:

- *Test Case Complexity (Tcplx).*

It is the measure of how difficult and complex to run the test case during testing process. It is also refers to the effort needed to execute the test cases. Weightage is assigned by true or false conditions where true denotes 10 points and false denotes 0 points for representing the complexity of test cases.

- *Test Impact (TI).*

It is based on impact on test cases during the testing of software. This factor helps to assess the importance of test cases to determine if test cases are not executed. Weightage can be given in three states such as high, medium, low where high denote 8to10 points, medium denotes 4to7 and low denote 1to3 points in scale assignment.

B) *The Proposed Prioritization Algorithm*

Values for all the 10 factors are assigned for each test case during test design analysis phase and evolve continually during software development process. We can compute weighted prioritization value (WPV) for each test case as follows:

$$WPV = \sum_{i=1}^{10} (PF \text{ value}_i * PF \text{ weight}_i) \dots (4)$$

Where,

- WP V is weight prioritization for each test case calculated from 10 factors
- PF value_i is a value assigned to each test case
- PF weight_i is a weight assigned for each factor

The computation of WPV for a requirement is used in computing the Weighted Priority(WP) for its associated test cases. Let there be n total requirements for a product and test case j maps to i requirements. Weighted Priority (WP) is calculated as follows,

$$WP_j = (\sum_{x=1}^i PFV_x / \sum_{y=1}^n PFV_y) \dots (5)$$

By calculating these values we can prioritize the test cases based on WPV and WP for each and every test case in the test suite.

Now we introduce the proposed technique in an algorithmic form here under:

Input: Test Suite *T* and Factor values for each test case

Output: Prioritized Test Suite *T'*

Algorithm:

Step 1. Begin

Step 2. set *T'* empty

Step 3. for each *t* ∈ *T* do

Step 4. Calculate Weighted prioritization value (WPV) using eqn(4)

Step 5. calculate Weighted Priority(WP) using eqn (5)

Step 6. end for

Step 7. Sort *T* in descending order based on the values of WP for each test case.

Step 8. Let *T'* be *T*

Step 9. end

IV. VALIDATION METRICS:

Validation of the proposed prioritization algorithm is carried out through three different validation metrics. First validation metric is based on the severity of fault detected for the product/software and the second metric is number of acceptable test cases based on size of prioritized test cases and third metric is based on total time taken for prioritization of test cases. We present these validation metrics in the following sections:

A. Defect Severity:

Efficiency of testing process can be improved by focusing on the test case that contains high number of severe faults. So, each fault severity value is assigned based on impact of the fault on the product. Severity value is assigned in 10 point scale assignment as shown below.

Very high severe	: SV of 32
High Severe	: SV of 16
Medium Severe	: SV of 8
Less Severe	: SV of 4
Least Severe	: SV of 2

Total Severity of Fault Detected (TSFD) is the summation of severity values of all faults known in a product. *F* number of faults identified for a product, TSFD can be computed as follows,

$$TSFD = \sum_{i=1}^f sv_i \dots (6)$$

Average severity of faults detected (ASFD) can be computed with *f* defects as,

$$ASFD = (\sum_{j=1}^f sv_j / TSFD) \dots (7)$$

B. Acceptable Test Case Size:

We propose to use the number of acceptable test cases is a validation metric because the size of prioritized test cases has an impact on the time, cost consumption and effort during the execution of regression testing. This metric is the number of acceptable test cases expressed in percentage as follows:

$$\%Size = (\text{size of each test case} / \text{total test case size}) * 10 \dots (8)$$

Where:

- %size is the size of acceptable test cases expressed in percentage
- Size of each test case is the no of test cases generated by each method excluding low priority test cases.
- Total test case size is the total number of test cases used in the experiment.

C. Total Prioritization Time

This is the total number of times the prioritization methods are run in the experiment. This is the time related metric used during pre-process and post-process of a test case prioritization.

$$TPT = T_{\text{Assigning weight}} + T_{\text{TCP Run}} \dots (9)$$

Where:

- TPT is the total time consumed in running the prioritization methods
- $T_{\text{Assigning weight}}$ is the total amount of time consumed in assigning weights, and computing weight prioritization values
- $T_{\text{TCP Run}}$ is the total time to run the test case prioritization methods including ordered test cases.

V. EXPERIMENTATION AND ANALYSIS:

The proposed test case prioritization algorithm was implemented in the working platform of JAVA (version JDK 1.6). Here we can use the student application projects for regression testing. In this example a test suite has been developed which consisting of 10 test cases and it covering a total of 5 faults. The regression test suite *T* contains 10 test cases with default ordering {T1, T2, T3, T4, T5, T6, T7, T8, T9, T10}. The faults covered by each test case and execution time required by each test case are shown in the table 1,

Table 1: Test Cases with faults covered and execution time for student project

Test Cases	Fault Covered	Execution Time(units)
T1	2,1,3	10.3
T2	4,1	3.7
T3	4,5	6.5
T4	2,3,5	8.2
T5	5,2,4,3	14.8
T6	4,2,5	16.4
T7	1,2,3	20.9
T8	1,5,3,4	5.2
T9	3	7.1

The evaluation of proposed approach is carried out by assigned by severity values and the Table 2 shows the number of defects identified by each test case, and the total time taken to detect the defects and severity values against those faults.

Table 2: Time taken to find defects and the severity value

Test Case/ fault	T1	T2	T3	T4	T5	T6	T7	T8	T9
F1	x	x					x	x	
F2	x			x	x	x	x		
F3	x			x	x		x	X	x
F4		x	X		x	x		X	
F5			X	x	x	x		X	
No.of.faults	3	2	2	3	4	3	3	4	1
Time(MS)	10	4	7	8	15	16	21	5	7
Severity Value	16	20	7	13	32	6	10	8	4

Execute the test case first which has highest severity values. The total severity becomes for the test cases as given below: T1=16, T2=20, T3=7, T4=13, T5=32, T6=6, T7=10, T8=8, T9=4. These severity values assign to these detected faults are shown in table 2. According to our approach test cases will be executed in the order: T5-T2-T1-T4-T5-T7-T8-T7-T9.

Table 3: PFV range for test cases and Defect Impact

PFV Values	Project 1		Project 2	
	DI	APFV	DI	APFV
1-3	12	10	4.5	14
3-5	23.5	12.4	19.8	12.6
5-7	32.8	35.2	27.5	27.8
7-10	45.2	36.8	39.2	30.5

The analysis of PFV and Defect Impact of two projects is shown in above table. PFV is calculated from all the test cases taken for a project are grouped into four factors. This result shows that the higher percentage of defect originates from the test cases with PFV values range of 5 or higher.

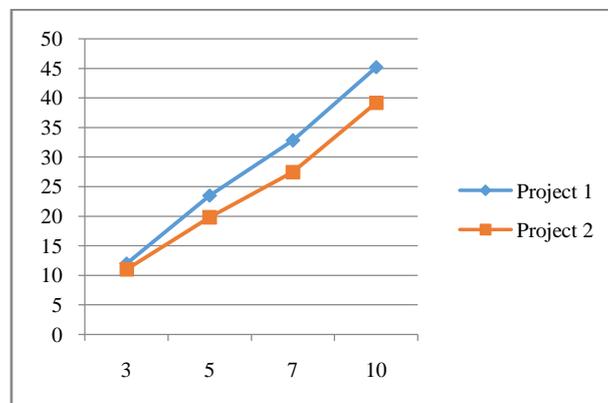


Figure 1: PFV vs. Defect Impact Range

Fig 1 evident that the test cases with higher DI originate with higher range of PFV.

A. Effectiveness of the proposed Technique

Here we can compare the effectiveness of the proposed technique by the three validation metrics by means of (1) random order execution (2) ordered execution (based on the proposed approach). Ten test cases are taken for comparison both in prioritized and in random order. Fig 1 shows that more number of defects can be detected in execution of test cases.

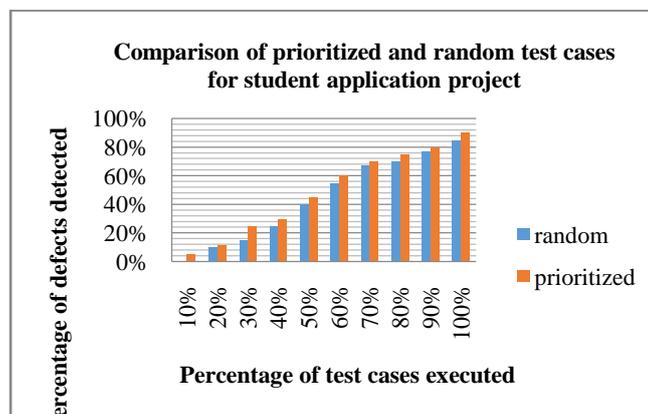


Figure 2: TSFD is higher for prioritized test case reveals more defects

The evaluation results for test case prioritization technique are shown in the figure 2. Validation metrics are Defect severity, acceptable test case size, total prioritization time for both random and prioritization time is evaluated and shown in the fig2.

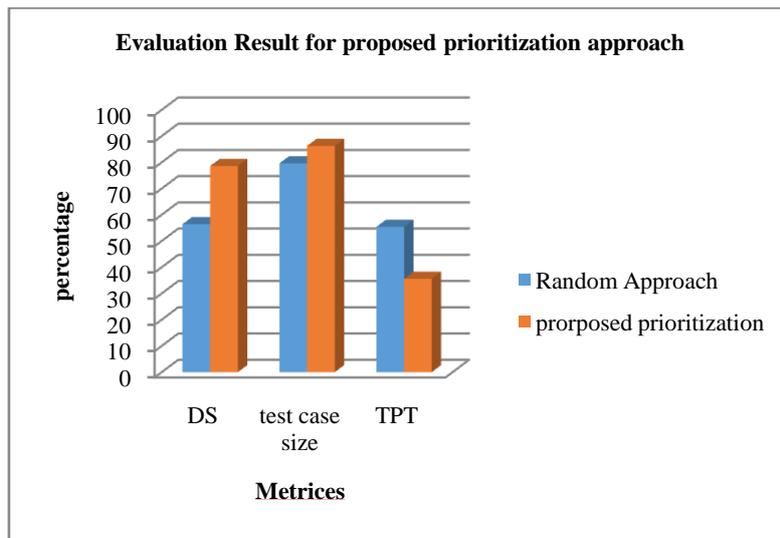


Figure 3: Evaluation result of test case prioritization methods

VI. CONCLUSION

In this paper, we propose a new prioritization technique for prioritizing system level test cases to improve the rate of fault detection for regression testing. Here we propose new practical set of weight factors used in the test case prioritization process. The new set of factors is composed of four groups: (1) Time Factor (2) Defect Factor (3) Requirement Factor (4) Complexity Factor for regression test cases. The proposed prioritization algorithm is validated by three validation metrics: (1) Defect Severity (2) Acceptable Test Case Size (3) Total Prioritization Time. Experimental Results shows that proposed technique leads to improve the rate of fault detection in comparison with random ordered test cases and reserves the large number of high priority test with least total time during a prioritization process.

REFERENCES

- [1] Elbaum, S., Malishvesky, A.G., Rothermel, G., 2002. Test case prioritization: a family of empirical studies. *IEEE Transactions on Software Engineering* 28 (2), 159–182.
- [2] Gregg Rothermel, Roland H. Untch, Chentun Chu and Mary Jean Harrold, “Prioritizing Test Cases for Regression Testing,” *IEEE Transactions on software Engineering*, VOL. 27 NO.10, October 2001.
- [3] Zheng Li, Mark Harman, and Robert M. Hierons, “Search algorithm for Regression Test Case Prioritization,” *IEEE Transactions on Software Engineering*, Vol. 33, No.4, April 2007.
- [4] Kim, J.-M., Porter, A., 2002. A history-based test prioritization technique for regression testing in resource constrained environments. In: *Proceedings of the 24th International Conference on Software Engineering*, pp. 119–129.
- [5] B. Qu, C. Nie, B. Xu, X. Zhang, Test case prioritization for black-box testing, in: *Proc. 31st Annual Int’l Computer Software and Applications Conf.*, vol. 1, 2007, pp. 465–474.
- [6] H. Park, H. Ryu, J. Baik, Historical value-based approach for cost-cognizant test case prioritization to improve the effectiveness of regression testing, in: *Proc. of the 2nd Int’l Conf. Secure System Integration and Reliability Improvement*, 2008, pp. 39–46.
- [7] Yu-ching Hung, kaun-Li Peng, Chin-Yu Haung, A History based cost cognizant test case prioritization technique in regression testing, in: *journals of system and software* 85(2012)
- [8] Harrold, M.J., Gupta, R., Soffa, M., 1992. An approach to regression testing using slicing. In: *IEEE-CS International Conference on Software Maintenance*, pp. 299–308.
- [9] Harrold, M.J., Rothermel, G., 1997. Aristotle: a system for research on and development of program analysis based tools. Technical Report OSU-CISRC-3/97-TR17, Ohio State University.
- [10] Jeffrey, D., Gupta, N., 2006. Test case prioritization using relevant slices. In: *Proceedings of 30th Annual International Computer Software and Applications Conference (COMPSAC 2006)*, Chicago, USA, September 18–21.
- [11] Agrawal, H., Horgan, J.R., Krauser, E.W., London, S., 1993. Incremental regression testing. In: *Proceedings of the IEEE Conference on Software Maintenance*, pp. 348–357.
- [12] Malishevsky, A.G., Ruthruff, J.R., Rothermel, G., Elbaum, S., 2006. Cost-cognizant Test Case Prioritization Technical Report TR-UNL-CSE-2006-0004. University of Nebraska-Lincoln.
- [13] G. Rothermel, M.J. Harrold, J. Ostrin, C. Hong, An empirical study of the effects of minimization on the fault detection capabilities of test suites, in: *Proc. of the Int’l. Conf. on Soft. Maintenance*, 1998, pp. 34–43.