Additional Fault Detection Test Case Prioritization

Ritika Jain[1] and **Neha Agarwal**[2]

Submitted in April, 2013; Accepted in August, 2013

Abstract - Regression testing is used to confirm that previous bugs have been fixed and that new bugs have not been introduced. Thus regression testing is done during maintenance phase and applied whenever a new version of a program is obtained by modifying an existing version. To perform a regression testing a set of new test cases and old test cases that were previously developed by software engineers are reused. This test suite is exhaustive in nature and it may take long time to rerun all test cases. Thus regression testing is too expensive and the number of test cases increases stridently as the software evolves. In present work, an additional fault detection test case prioritization technique is presented that prioritizes test cases in regression test suite based on number of concealed faults detected by test cases. Both noncost cognizant and cost cognizant prioritization of test cases have been performed using proposed technique and efficiency of prioritized suite is assessed using APFD and APFDc metric respectively.

Index Terms – APFD, APFDc, Cost-cognizant, Regression testing, Test case prioritization.

1.0 INTRODUCTION

A software product goes through requirement elicitation phase, design phase, implementation phase and testing phase before being fully operational and ready for use [1]. At the coding time developers often saves the set of test cases to test the module written and to reuse them later whenever software undergoes changes. When testing phase starts testing team creates a separate test suite, test environment and test oracle to validate the whole software system against customer satisfaction. Most of the time it is assumed that the software life cycle end after its delivery. But a long lasting maintenance phase starts after the deployment of the software to the customer's site. During this long life software evolves through numerous modifications and additions based on faults, change of user requirements, change of working platform or environments, change of government policies, and so forth. With the evolution of software product, maintaining its quality becomes more difficult and harder due to its numerous released versions, which goes on incrementing with every new set of changes to the previous version. These modifications into the existing software or addition of new features to the software may create new faults or may cause it to work improperly.

[1, 2]*Department of Computer Science and Engineering, Amity School of Engineering and Technology, Amity University, Noida-201303, India*
E-mail: [1]jritika27@gmail.com and [2]nagarwal2@amity.edu

Thus assuring the quality of software product along with these modifications and additions is the challenging task of maintenance phase. Sometimes the quality may become worse than before. On the other hand, users hope that the new version of the software should be easy to use, has more features, and has better quality than before.

Regression testing is used to confirm that fixed bugs have been fixed and that new bugs have not been introduced. Regression testing refers to that portion of test cycle in which a program P' is tested to ensure that not only does the newly added or modified code behaves correctly, but also the code that carried over unchanged from the previous version P continues to behave correctly. Thus regression testing is done during maintenance phase and applied whenever a new version of a program is obtained by modifying an existing version. Regression testing is sometimes referred to as "program revalidation". The term "corrective regression testing" refers to regression testing of a program done after making corrections to the previous versions. Another term "progressive regression testing" refers to a regression testing of a program done when new features are added to the previous version. To perform a regression testing a set of new test cases and old test cases that were previously developed by software engineers are used. This test suite is exhaustive in nature and it may take long time to rerun all test cases. Thus regression testing is too expensive and the number of test cases increases stridently as the software evolves.

Researchers [2-3] have provided effective regression testing techniques. The simplest one is to reuse all test cases that were run before the modification of previous version of the software. But it might waste time and resources due to execution of unnecessary tests. Therefore it is desirable to run only a subset of test suite, which can be chosen by using regression test selection techniques. The drawback of test subset selection techniques is that some important and crucial test cases might be evaluated as worthless and might not be selected for execution, which might cause some effected portion of the software to be remained untested. Another approach is to permanently remove the number of test cases from the test suite by eliminating redundant test cases and thus reducing the test suite size, which can be accomplished by using test suite reduction techniques. The downside of this is that it might degrade the fault detection capability with the reduction of test suite size. The above discussed problems can be solved by test case prioritization techniques, in which test cases in a test suite are rearranged in an attempt to make sure that faults get revealed at earlier stage of testing process. Test case prioritization techniques schedule test cases in an order such that those with higher priority, according to some objective criteria, are executed before than those with lower priority, to meet some performance goal. Test case prioritization can be

used in conjunction with test subset selection and test reduction techniques to prioritize test cases in selected or minimized test suite [4-6].

In this paper, a new test case prioritization algorithm has been proposed that prioritizes the test cases in the test suite based on the maximum number of faults detected which has not been revealed by any other test case executed so far. In first experiment, test cost and fault severities are considered as unity and then test cases are prioritized using proposed algorithm. The efficiency of prioritized suite is then measured using APFD metric. In second experiment, test cost and fault severities are incorporated while prioritizing test cases and measured using APFDc metric. In the end experimental results are analyzed and compared.

The rest of this paper is structured in the described format. In Section 2, a brief description of test case prioritization technique is given followed by literature survey in Section 3. The new algorithm to test case prioritization is presented in Section 4. The experimental studies along with the obtained results and their analysis are given in Section 5. Finally, some conclusions are drawn in Section 6.

2. TEST CASE PRIORITIZATION

Test case prioritization problem is defined as finding a permutation of test cases to maximize an objective function, which reflects the prioritization goal. The formal definition of this problem can be expressed as follows [3, 7-9].

2.1 Problem Statement

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

Problem: Find $T' \in PT$ such that $(\forall T'')\ (T'' \in PT)\ (T'' \neq T')\ [f(T') \geq f(T'')]$

Here, PT represents the set of all possible prioritizations (orderings) of T, and f is a function that, applied to any such ordering, yields an award value for that ordering. The definition assumes that higher award values are preferred over the lower ones.

2.2 Measuring Efficiency

In order to measure the effectiveness of test case ordering obtained through test case prioritization technique, a metric APFD (Average Percentage of Fault Detected) has been defined in the literature [3]. This metric measures average percentage of faults detected (rate of fault detected) against average percentage of test suite executed. Its value ranges from 0% to 100%. Higher APFD value reveals higher rate of fault detection and lower value reveals lower rate of fault detection. This metric provides a mean to compare efficacies of various prioritization techniques. In a formulaic presentation of the APFD metric, let T be a test suite containing n sequential test cases, and F be a set of m faults revealed by T. Let T' be some ordering of T. Let TF_i be the first test case in T' that reveals fault i. The APFD for T' is:

$$APFD = 1 - \frac{TF_1 + TF_2 + TF_3 + \cdots + TF_n}{nm} + \frac{1}{2n} \quad (1)$$

The limitation of APFD metric is that it treats all test cases with equal cost and all faults with equal severity. However, practically these factors possess distinct value for different test case and drastically affect the effectiveness of test case ordering. In such cases, the APFD metric provides unsatisfactory results. To triumph over the weakness of APFD metric a new metric APFDc (Average Percentage of fault Detected per Cost) has been suggested in the literature [3]. This metric measures average percentage of test case cost incurred against average percentage of fault severity detected. The APFDc metric can be quantitatively described as follows: let T be a test suite containing n test cases with cost $t_1, t_2, \ldots, t_n$. Let F be a set of m 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 an ordering T' of T that reveals fault i. The weighted (cost-cognizant) average percentage of faults detected during the execution of test suite T' is given by the equation:

$$APFD_c = \frac{\sum_{i=1}^{m}(f_i \times (\sum_{j=TF_i}^{n} t_j - \frac{1}{2} t_{TF_i}))}{\sum_{j=1}^{n} t_j \times \sum_{i=1}^{m} f_i} \quad (2)$$

3. LITERATURE SURVEY

Researchers [7, 9-10] have provided variety of test case prioritization techniques. Most of these techniques prioritize test cases based on their coverage information such as statement coverage, branch coverage, loop coverage, function coverage, condition coverage, and fault detected. Different prioritized suite of test cases are then obtained based on different coverage techniques and evaluated using APFD metric in order to contrast them. However, these code coverage based prioritization techniques have two limitations.

First, these techniques consider the value of test cost and fault severity as equivalent. On the contrary, factors allied to test cost and fault cost radically affect the ordering of test cases in prioritized suite and so its efficacy. As a consequence, researchers work out different ways to compute test cost and fault cost and provide cost cognizant test case prioritization techniques [2-3, 8, 11] that include these factors while prioritizing test cases. The efficacy of cost cognizant test case prioritization techniques can be measured through APFDc metric.

Literature [12-13] also proposed historical value based approach for cost cognizant test case prioritization in which a historical value model is used to calculate historical value of test cases based on the historical test case cost and the historical fault severities.

Second limitation is that, testers needed source code to assess coverage information of test cases and if source code is not available then applying this technique will be very difficult. In order to beat this difficulty, history-based test prioritization techniques were proposed and studied [14-15] that uses historical execution data of test cases in order to determine their priority. In addition, some researchers [16-17] had

employed genetic algorithm to history–based test case prioritization. They collect data such as test cost, fault severities, and detected faults of each test case from the latest regression testing and apply genetic algorithms to acquire better prioritized suite.

4. PROPOSED TECHNIQUE

In present communication, a new regression test suite prioritization algorithm is presented, that prioritizes the test cases in the regression test suite with the goal of finding the maximum number of faults at the early phase of the testing process. It is presumed that the desired execution time to run the test cases, the faults they reveal, and the fault severities are known in advance. This technique considers total number of concealed faults detected by each test case, to prioritize them.

First the total number of faults detected by each test case is found. From this, the one which detects maximum faults is selected and then the fault coverage data of each unselected test case is adjusted to indicate their new fault coverage data (faults detected by each test case that are not yet discovered). Then the test case which covers maximum faults is selected. If there is more than one test case which covers maximum number of faults then choose them randomly and again adjust the fault coverage data of unselected test cases. This process is repeated until all faults have been covered. When all faults have been covered, same process is repeated for the remaining test cases. To make this technique accustomed to the situation where test costs and fault severities vary, instead of summing the number of new faults covered by a test case t to calculate the worth of t, the number of new faults f covered by t is multiplied by the criticality-to-cost adjustment g(criticality$_t$;cost$_t$) for t. The notion behind the use of this computation is to reward those test cases that have greater cost adjustments when weighted by the additional faults they cover.

Algorithm 1: when test cost and fault severity are unity

Input: Test suite T, fault trace history F_h
Output: Prioritized test suite T'
1: begin
2: set T' empty
3: initialize values of vector F_{cov} as “uncovered"
4. While T is not empty do
5: for each test case t Є T do
6: calculate total number of faults f covered by t using F_h and F_{cov}
7: end for
8: select test t that cover maximum number of faults
9: append t to T'
10: update F_{cov} based on the faults covered by t
11: if all faults has been covered
12: initialize values of vector F_{cov} as “uncovered"
13: end if
14: remove t from T
15: end while
16: end

Algorithm 2: when test cost and fault severity are different

Input: Test suite T, fault trace history F_h, test criticalities T_{crit}, test costs T_{cost}
Output: Prioritized test suite T'
1: begin
2: set T' empty
3: initialize values of vector F_{cov} as “uncovered"
4. While T is not empty do
5: for each test case t Є T do
6: calculate total number of faults f covered by t using F_h and F_{cov}
7: calculate award value of t as f * g (criticality$_t$; cost$_t$) using T_{crit} and T_{cost}
8: end for
9: select test t with the greatest award value
10: append t to T'
11: update F_{cov} based on the faults covered by t
12: if all faults has been covered
13: initialize values of vector F_{cov} as “uncovered"
14: end if
15: remove t from T
16: end while
17: end

F_{cov} is a vector having values “covered" or “uncovered" for each fault in the system. The vector record the faults that have been covered by previously selected test cases. T_{crit} is the criticality of test case measured by summing the severities of all faults covered by test case.

4.1 Comparative Techniques

No prioritization. As an experimental control, one comparator technique that we consider is no prioritization, where no prioritization technique is applied and test cases are executed in sequential order.

Random ordering. Another comparator technique that we consider is the random ordering of the test cases in the test suite.

4.2 Estimating Test Cost and Fault Severity

For cost cognizant prioritization it is required to obtain cost of each test case and severity of faults each test case reveal. [3, 13] makes available some measures to compute test cost and fault severity. Test costs are greatly diversified in software testing. Depending on the criteria, a test cost can be computed over several factors such as machine time, human time, test case execution time, monetary value of the test execution, and so forth. Similarly, fault severity can also be measured by depending upon criteria such as test criticality (the criticality of the test case that detects a fault) and function criticality (the criticality of the function in the code that is covered by the test case).In our approach, test cost is refined as the test case execution time of a test case and Fault severity is refined to test case criticality, which is devoted to each test case by summing up the severities of each fault that test case reveal.

4.3 Award Value Calculation

In cost cognizant prioritization the test cases are prioritized based on award value. The award value of a test case is calculated using the formula

Award value = $f \times g(\text{criticality}_t, \text{cost}_t)$

Where criticality_t is the total severity of faults detected by test case t, cost_t is the cost of test case t, and g is a function that maps the criticality and cost of t into a value. (Function g simply divides criticality_t by cost_t.). Greater the award value more will be the chances of the test case to be selected for execution.

5. RESULTS AND DISCUSSION

Table 1 shows test cost and fault severity of test cases and faults respectively [11]. The proposed technique is examined on data given in Table 1 and comparative analyses are drawn.

Fault / test case	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	Fault severity
F1								$	$		6
F2		$	$		$						6
F3				$		$				$	6
F4		$	$								10
F5								$			8
F6								$	$		10
F7				$	$		$				4
F8	$					$					20
F9				$		$				$	12
F10	$							$			6
Number of faults	2	2	2	3	2	3	1	4	2	2	
Time	9	8	14	9	12	14	11	10	10	13	

Table 1: Time taken to find out the fault and the severity value [11]

5.1 Experiment 1

Test cases are prioritized based on the total number of concealed faults detected by each test case. Here test cost and fault severity is considered as unity. From Table 1 as it can be seen, test case T8 has revealed maximum number of faults, thus the first test case to be executed in prioritized suite is T8. The new fault coverage data of remaining test cases adjusted to T8 are shown in Table 2.

Fault / test case	T1	T2	T3	T4	T5	T6	T7	T9	T10
F2		$	$		$				
F3				$		$			$
F4		$	$						
F7				$	$		$		
F8	$					$			
F9				$		$			$
Number of faults	1	2	2	3	2	3	1	0	2

Table 2: Fault coverage data of test cases adjusted to test case T8

Now there are two test cases that expose three new faults. Next test case to be executed is T4. In this way, the prioritized suite is produced by applying the additional fault detection technique. Table 3 shows test suite prioritized order both for comparative techniques and proposed technique.

Prioritization technique	Prioritization order
No prioritization	T1-T2-T3-T4-T5-T6-T7-T8-T9-T10
Random ordering	T5-T3-T9-T1-T8-T6-T2-T10-T7-T4
Additional fault detection	T8-T4-T2-T1-T6-T3-T9-T5-T10-T7

Table 3: Prioritization order based on additional fault detection without considering cost

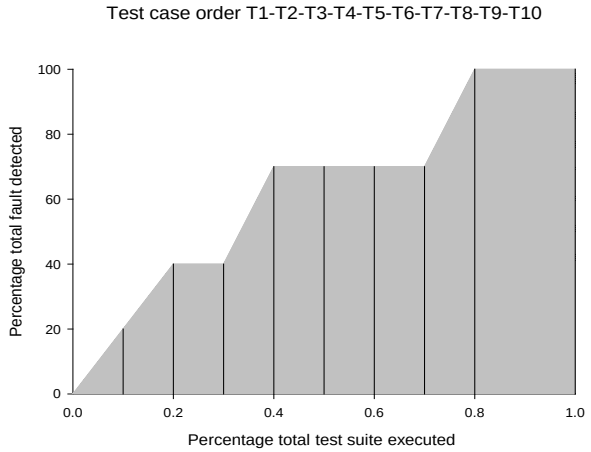
The efficiency of this prioritized order is measured through APFD metric and its value is given in Table 4.

Prioritization technique	APFD (%)
No prioritization	53
Random prioritization	60
Additional fault detection	75

Table 4: APFD Value of prioritized suite

It is seen that proposed prioritization technique increases the rate of fault detection capability of regression test suite upto 75% from 53% when there is no prioritization. The APFD value of prioritization order obtained through proposed technique is also greater than random ordering which makes it clear that random prioritization is never reliable. The proposed technique is also compared with the technique given by Kavitha and Sureshkumar [11]. It can be presented that the APFD value obtained by proposed technique is 75% whereas it is reported to be 70% for the same test data by Reference [11]. APFD graphs of unprioritized suite, random ordered suite, and additional fault detection prioritized suite is demonstrated in Figure 1a, 1b, and 1c respectively. The horizontal line in the graph represents average percentage of test suite executed and

vertical line represents average percentage of total faults detected.

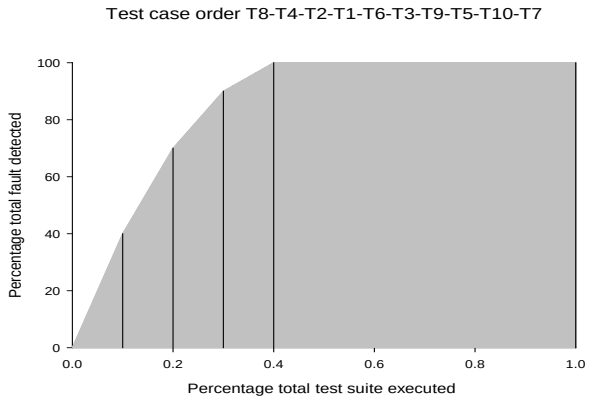


APFD = 53%

(a)

APFD = 60%

(b)



APFD = 75%

(c)

Figure 1: APFD graph of (a) unprioritized test suite, (b) random ordering test suite and (c) test suite from proposed technique

5.2 Experiment 2

Since test cost and fault severity greatly varies for each test case and fault, ignoring them can never produce appropriate and satisfactory results. Thus test cost and fault severity is integrated with test cases and faults respectively to produce a prioritized suite that detects more and more severe faults by incurring less cost to execute test cases. Test cases are prioritized based on award value of each test case. Efficiency of the prioritized test suite is measured through APFDc metric. Prioritized suite that is obtained by applying algorithm 2 on the comparative techniques and on data shown in Table 1 is presented in Table 5.

Prioritization technique	Prioritization order
No prioritization	T1-T2-T3-T4-T5-T6-T7-T8-T9-T10
Random ordering	T5-T3-T9-T1-T8-T6-T2-T10-T7-T4
Additional fault detection	T8-T6-T2-T4-T1-T9-T10-T3-T7-T5

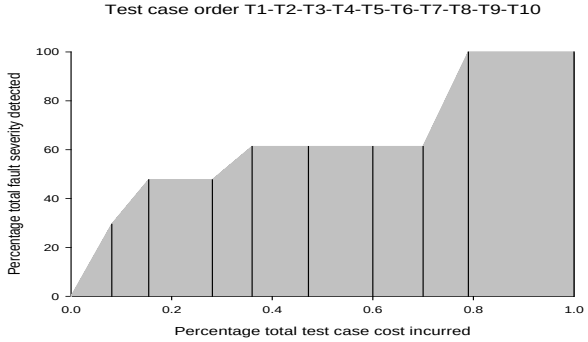
Table 5: Prioritization order based on additional fault detection considering cost

For these prioritized order their efficiency is measured through APFDc metric and its value is given in Table 6.

Prioritization technique	APFDc (%)
No prioritization	68.24%
Random prioritization	63.95%
Additional fault detection	85.64%

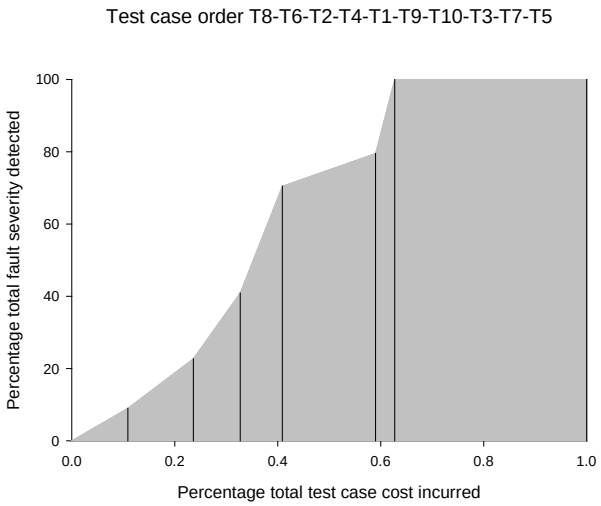
Table 6: APFDc value of prioritized suite

It can be observed from Table 6 that the efficiency of random ordering can sometimes be less than unprioritized test suite. Proposed prioritization technique shows highest rate of fault exposing potential (85.64%). It is also observed that cost cognizant test case prioritization has considerable higher fault exposing rate than noncost cognizant prioritization technique. APFD graph of unprioritized suite, random ordered suite, and additional fault detection prioritized suite is demonstrated in Figure 2a, 2b, and 2c respectively. The x-axis represents weighted "average percentage of test cost incurred" and y-axis represents weighted "average percentage of fault severity detected".



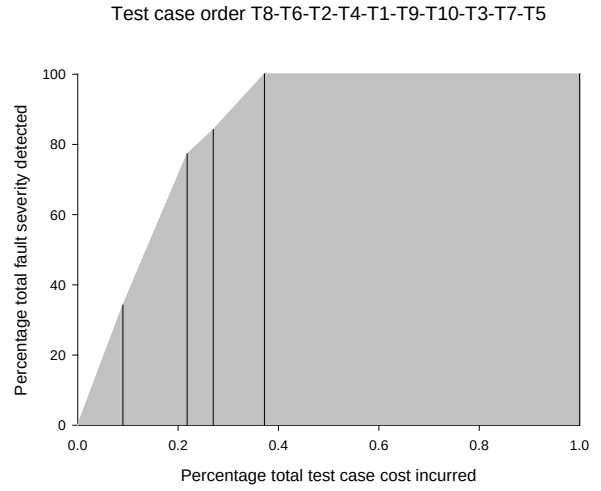
APFDc = 68.24%

(a)



APFDc = 63.95

(b)



APFDc = 85.64

(c)

Figure 2: APFDc graph of (a) unprioritized test suite, (b) random ordering test suite and (c) test suite from proposed technique

6. CONCLUSIONS

In present communication, a new prioritization technique for Regression testing is presented that prioritizes test cases based on exposure of undetected faults by each test case to improve the rate of fault detection. The results show that the proposed technique leads to improve the rate of detection of severe faults at early stage in comparison to nonprioritized order and random order of test cases. When test cases are prioritized without considering the cost of tests and severity of faults, prioritized suite of proposed technique gives 75% APFD value, which is very large as compared to no prioritization and random prioritization. When the same technique is integrated with test cost and fault severities, prioritized order of proposed technique gives 85.64% APFD value which is not only larger than comparative techniques but also larger that APFD value of noncost cognizant prioritized suite of proposed technique. It is also observed that the number of test cases required to find all faults is less in case of proposed prioritization technique. The important feature of this technique is that it exposes abundance amount of severe faults in short duration of test suite execution.

REFERENCES

[1]. K.K. Agarwal, and Y. Singh, "Software engineering," IIIrd ed, New age international publishers, 2008.

[2]. P.R. Srivastava, "Test case prioritization," *Journal of Theoretical and Applied Information Technology*, vol. 4 (3), pp. 178-181, 2008.

[3]. A.G. Malishevsky, G. Rothermel, and S. Elbaum, "Cost-cognizant test case prioritization," *Technical Report TR-UNL-CSE-2006-0004*, Department of Computer Science and Engineering, University of Nebraska-Lincoln, Lincoln, Nebraska, U.S.A., 12 March 2006.

[4]. K.K. Aggrawal, Y. Singh, A. Kaur, "Code coverage based technique for prioritizing test cases for regression testing," *ACM SIGSOFT Software Engineering*, vol. 29, no. 5, Sep. 2004.

[5]. R. Malhotra, A. Kaur, Y. Singh, "A regression test selection and prioritization technique," *Journal of Information Processing Systems*, vol.6, no.2, June 2010.

[6]. G. Rothermel and M.J. Harrold, "A safe, efficient regression test selection technique," *ACM Trans. Software Engineering and Methodology*, vol. 6, no. 2, pp. 173-210, Apr. 1997.

[7]. G. Rothermel, H. Roland H. Untch, Mary Jean Harrold, "Prioritizing Test Cases For Regression Testing", *IEEE Transactions on Software Engineering*, vol. 27, no. 10, October 2001.

[8]. S. Elbaum, A. Malishevsky, G. Rothermel, "Incorporating Varying Test Costs and Fault Severities into Test Case Prioritization," *Proc. 23rd International Conference on Software Engineering*, May 2001.

[9]. S. Elbaum, A. Malishevsky, G. Rothermel, "Test Case Prioritization: A Family of Empirical Studies, *IEEE Transactions on Software Engineering*, vol. 28, No. 2, February 2002.

[10]. G. Rothermel, R. Untch, C.chu, M. Harrold, "Test case prioritization:an empirical study," *Proc. International conference on Software Maintenance*, pp. 179-188, Aug. 1999.

[11]. R. Kavitha, N. Sureshkumar, "Test Case Prioritization for Regression Testing based on Severity of Fault", *International Journal on Computer Science and Engineering*, vol. 02, No. 05, 2010, 1462-1466.

[12]. H. Park, H. Ryu, J. Baik, "Historical value based approach for cost cognizant test case prioritization to improve the effectiveness of regression testing," *2nd*

International Conference on Secure System Integration and Reliability Improvement, pp. 39-46, July 2008.

[13]. P. Srivastava, "Performance Evaluation of Cost-cognizant Test Case Prioritization", *UACEE International Journal of Computer Science and its Applications*, Volume 1, Issue 1.

[14]. A. Khalilian, M. A. Azgomi, Y. Fazlalizadeh, "An improved method for test case prioritization by incorporating historical test case data", *Science of Computer Programming*, volume 78, issue 1, 1 November 2012, pages 93-116.

[15]. J. M. Kim, A. Porter, "A history based test prioritization technique for regression testing in resource constrained environment," *In Proc. of the International Conference on Software Engineering*, Orlando, USA, pp. 119-129, 2002.

[16]. Y.C. Huang, C.Y. Huang, J.R. Chang, T.Y Chen, "Design and Analysis of Cost-Cognizant Test Case Prioritization Using Genetic Algorithm with Test History," *IEEE 34th Annual Computer Software and Applications Conference*, 2010.

[17]. Y.C. Huang, K.L. Peng, C.Y. Huang, "A history-based cost-cognizant test case prioritization technique in regression testing," *The Journal of Systems and Software*, 85, 2012, 626– 637.