International Journal of Applied Information Systems (IJAIS) – ISSN : 2249-0868
Foundation of Computer Science FCS, New York, USA
Volume 6– No. 7, January 2014 – www.ijais.org

A New Effective Test Case Prioritization for Regression Testing based on Prioritization Algorithm

Thillaikarasi Muthusamy[1]
[1](Assistant Professor) Department of computer science and Engineering,Faculty of Engineering and technology
Annamalai University,Annamalai Nagar, Tamilnadu,India-608002

Seetharaman.K[2], Ph.D
[2](Associate Professor) Department of computer science and Engineering, Faculty of Engineering and technology
Annamalai University,Annamalai Nagar, Tamilnadu, India-608002

Abstract
Regression Testing is the process of executing the set of test cases which have passed on the previous build or release of the application under test in order to validate that the original features and functions are still working as they were previously. It is impracticable and in-sufficient resources to re-execute every test case for a program if changes occur. This problem of regression testing can be solved by prioritizing test cases. A regression test case prioritization technique involves re-ordering the execution of test suite to increase the rate of fault detection in earlier stages of testing process. In this paper, test case prioritization algorithm is proposed to identify the severe faults and improve the rate of fault detection. This proposed test case prioritization algorithm prioritizes the test cases based on four groups of practical weight factor such as: customer allotted priority, developer observed code execution complexity, changes in requirements, fault impact, completeness and traceability. The proposed prioritization technique is validated with three different validation metrics and is experimented using two projects. The effectiveness of proposed technique is achieved by comparing it with un-prioritized ones and by validation metrics.

Keywords
Regression Testing, Test case prioritization, Fault severity, Rate of fault detection.

1. INTRODUCTION
Software regression testing is an activity which includes enhancements, error corrections, optimization and deletion of existing features. These modifications may cause the system to work in-correctly. Therefore, Regression Testing becomes Necessary in software testing process. One of the methods for regression testing in which all the tests in the existing test bucket or suite should be re-executed. This is very expensive as it requires huge time and resources. Test case prioritization is the important technique carried out in regression testing. Prioritize the test cases depending on business impact, critical & frequently used functionalities. Selection of test cases based on priority will greatly reduce the regression test suite. In this paper we propose a new approach for test case prioritization for earlier fault detection in the regression testing process.

In this paper, we proposed a new approach to test case prioritization for quick fault detection based on practical weight factors. We have implemented the proposed technique using a banking application project and effectiveness is calculated by using APFD metric.

2. TECHNIQUES REVISITED
This section describes the test case prioritization techniques to be used in our empirical study are as follows:

Test case prioritization is an important regression testing technique Test case prioritization approaches typically sort existing test cases for regression testing according to attain performance goals.

Badhera et al.[1] presented a technique to execute the modified lines of code with minimum number of test cases. The test case prioritization technique organizes the test case in a test suite in an ordering such that fewer lines of code need to be re executed thus faster code coverage is attained which would lead to early detection of faults. Bixin Li et al.(2012) proposed an automatic test case selection for regression testing of composite service based on extensible BPEL flow graph.

B. Jiang et al. [2] proposed an ART-based test case prioritization uses the algorithm which accepts the test suite as input and produces the output in prioritized order of test cases. The basic idea behind is by building the candidate set of test cases which in turn picks one test case from the candidate set until all test cases have been selected.
Here two functions are used in this algorithm for calculating the distance between a pair of test cases and for selecting a test case from the candidate set. Calculation of distance is mainly based on code coverage data. Then we find a candidate test case is associated with the distance with the test cases that have been already selected.

Dr. ArvinderKaur and ShubhraGoyal [3] proposed a new genetic algorithm and prioritize regression test suite within a time constrained environment on the basis of total fault coverage. This algorithm is automated and the results are analyzed with help of Average Percentage of Faults Detected (APFD).

Hong Mei et al. [4] proposed a new approach for prioritizing test cases in the absence of coverage information which widely used in java programs under the JUnit framework. A new approach called JUPTA(JUnit test case Prioritization Techniques operating in the Absence of coverage information) which analyzes the static call graphs of JUnit test cases and estimate the ability of each test case to achieve code coverage and schedules the test cases in order based on those estimates.

H.Do et al. [5] presented the effects of time constraints on test case prioritization and find that constraints which alters the technique performance. They conducted three set of experiments which exhibits the time constraints. The experiment results show that the time constraint factor shows the significant role in determining the cost effectiveness and cost benefit trade-offs among the techniques. Next experiment replicates the first experiment, controlling for several threats to validity including numbers of faults present, and third experiment manipulates the number of faults present in programs to examine the effects of faultiness levels on prioritization and shows that faultiness level affects the relative cost-effectiveness of prioritization techniques.

Park et al. [6] introduced a cost cognizant model for the test case prioritization and fault severities revealed in the lack previous test execution does not significantly change form one release to another. Mohamed A Shameem et al. (2013) presented a metric for assessing the rate of fault dependency detection. This algorithm identifies the faults in earlier stages and the effectiveness of the prioritized test cases are compared with the non prioritized ones by Average Percentage Of Fault Detected (APFD).

M. Yoon et al. [7] proposed a method to prioritize new test cases by calculating risk exposure value for requirements and analyzing risk items based on the calculation to evaluate relevant test cases and thereby determining the test case priority through the evaluated values. Moreover, we demonstrate effectiveness of our technique through empirical studies in terms of both Average Percentage Of Fault Detected (APFD) and fault severity.

R. Abreu et al. [8] proposed a Spectrum-based multiple fault localization method to find out the fault location very clearly.R. Bryce et al. (2011) proposed a model which defines prioritization criteria for GUI and web applications in event driven software. The ultimate goal is to evolve the model and used to develop a unified theory of how all EDS should be tested.

R. Krishnamoorthi and S. A. Mary [9] presented a model prioritizes the system test cases based on six factors: customer priority, changes in requirement, implementation complexity, usability, application flow and fault impact. This prioritization technique is experimented in three phases with student projects and two sets of industrial projects and the results improved the rate of severe fault detection

S. Raju and G.V. Uma [10] introduced a cluster-based test case prioritization technique. By clustering test cases, based on their dynamic runtime behavior researchers can reduce the required number of pair-wise comparisons significantly. Researchers present a value-driven approach to system-level test case prioritization called the prioritization of requirements for test. In this approach, prioritization of test cases is based on four factors rate of fault detection, requirements volatility, and fault impact and implementation complexity.

The rest of this paper is organized as follows. In section three discusses about the proposed work. Section four discusses about the experimental results and analysis. Section five discusses about the discussions. And finally, section six consists of conclusion. References are given in last section.

3. PROPOSED WORK

This section, we briefly discuss about the prioritization factors.

3.1 Prioritization Weight Factors

Computation of proposed practical prioritization factors such as (1) customer allotted priority , (2) developer observed code execution complexity, (3) changes in requirements, (4) fault impact (5) completeness and (6) traceability , is essential for prioritizing the test cases because they are used in the prioritization algorithm. Weights are assigned to each test case in the software according to these factors. Then, test cases are prioritized based on the weights assigned.

3.1.1 Customer-Allotted Priority (CP)

It is a measure of the implication of a requisite to the customer. The values of each need are assigned by the customers. The values vary from 1 to 20, where 20are used to identify the highest customer priority. So, improving customer's fulfillment imposes the initial testing of the highest priority needs of the customers. Greater effort should be consumed in identifying faults and their impacts that take place on the execution path of program as these faults results in repeated failures. It has been proved that customer-Allotted value and satisfaction can be improved by fixing on customer needs for development.

3.1.2 Developer-observed Code Implementation Complexity(IC)

It is an individual measure of the complexity expected by the development team in implementing the necessity. First every necessity is evaluated. The developer assigns a value from 1 to20 on the basis of its implementation complexity and a higher complexity is implied by a larger value. Large number of faults that could be occurs in a requirement that has high implementation complexity.

3.1.3 Changes in Requirements (RC)

It is a degree assigned by the developer in the range of 1 to 20 for indicating the number of times a requirement is changed in the development cycle with respect to its origin date. The volatility values for all the needs are expressed on a 20-point scale is the need is altered more than 20 times. The number of changes for any requirement i divided to the highest number of changes for any requirement among all the project requirements yields the change in requirement *Ri* of that requirement i. If the ith requirement is changed M times and N is the maximum number of requirements then the requirement change of *i*, *Ri* can be calculated in Eqn(1) as

$$Ri = (M/N) \times 10 \qquad (1)$$

The errors introduced in the requirement level are approximated to 50% of all faults detected in the project. The change in requirements is the major factor attributable to the failure of the project.

3.1.4 Fault Impact of Requirements (FI)

It allows the development team to distinguish the requirement that had customer reported failures. Developers can recognize requirements that are expected to be error free by using the prior data collected from older versions as a system evolves to several versions. The number of in-house failures and field failures determine the fault impact of requirements. It is measured for those that have been in a released product. It is proved field failures are caused more likely to be fault prone modules than modules that are not fault prone.

3.1.5 Completeness (CT)

This part indicates what is needed as per the requirement for a function to be executed, the rate of success, the limitations to be followed for the function is to be executed and any limitation which manipulate the expected solution for example the boundary constraints. The consumer assigns value from 1 to 20.When the condition is selected for reuse after scrutinizing the completeness of each requirement into consideration, customer satisfaction like stronghold of the software response to the user request can be enhanced.

3.1.6 Traceability (TR)

Relation between requirement and assessment can be calibrated by means of Traceability. Defining whether a requirement is properly tested is cumbersome for evaluators. If the test cases are not concerned to individual requirement, the common problem reported is scarcity of traceability, hence poor traceability leads to failure and going beyond the desired limit of the project. It is executed by undergoing précised way rather than a conventional process. Most of the minor cases for software failures are identified due to lack of traceability. Requirement traceability may be defined as ability to monitor life of requirement in either ways i.e. from the inception through construction and specification and for its subsequent execution and usage through steps of continuous advancement and recurrence in any of the stages. The evaluator allots value in the range from 1 to 20, after assessing individual requirement for the concerned traceability and the standard of software can be improved by opting the traceability of the requirement into consideration is chosen for subsequent usage.

3.2 Proposed Prioritization Algorithm:

Values for all the 6 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 i shown in Eqn(2)

$$\text{WPV}=\sum_{i=1}^{10}(PFvalue_i * PF\ weight_i) \quad (2)$$

Where,WPV 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 in Eqn(3) as

$$\text{WP}_{j=}\ (\sum_{x=1}^{i} PFV_x / \sum_{y=1}^{n} PFV_y) \quad (3)$$

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. **Fig. 1** shows, which explains the overview for the proposed prioritization approach which comprises of prioritization factor values for each test case normalized to 20 values and we can prioritize those test cases based on weighted priority value then produces the prioritized test suite.

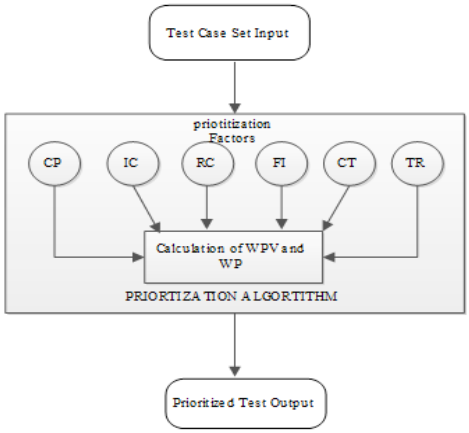
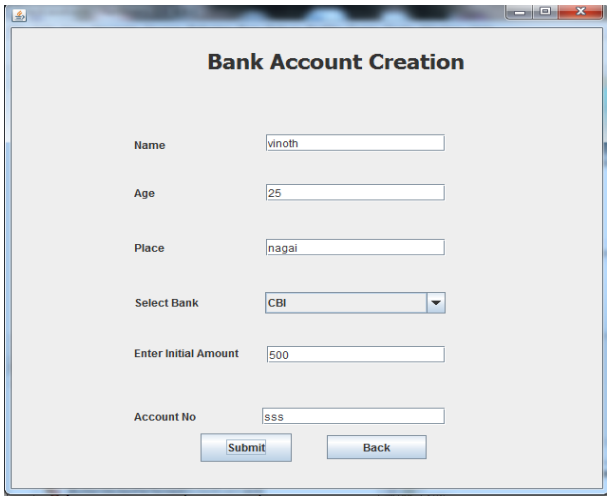


Fig. 1 Overview of the implementation of proposed technique

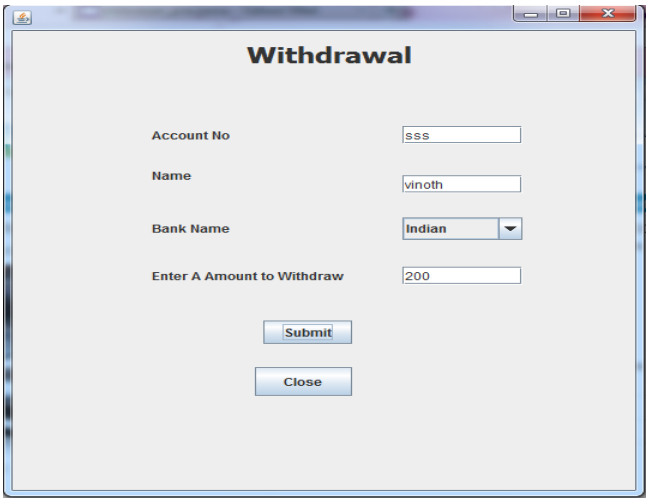
Now we introduce the proposed technique in an algorithmic form here under: This algorithm calculates WPV (weighted priority value) and WP (Weighted Priority) for every test cases which takes into the account of un-prioritized test input. Then any sorting algorithm like quick sort or heap sort can be implemented to sort the WP values in descending order.

3.2.1. Algorithm

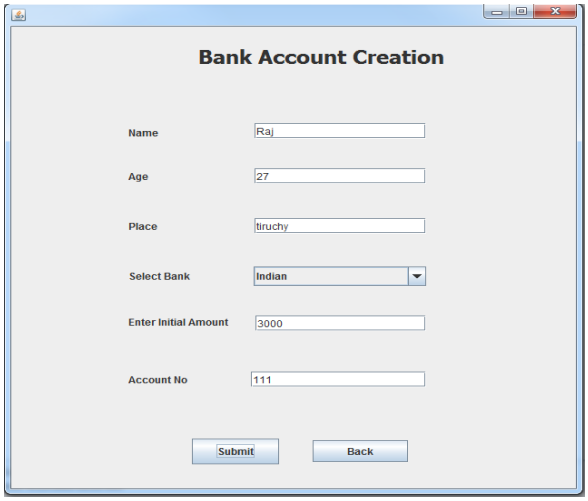
Input: Test Case Set (denoted as TS)
Output: Prioritized Test Suite (denoted as PS)
General Process:
Begin
For each test case t in TS
    Calculate WPV for t
End for
While TS is not empty do
    Calculate WP in TS
End While
Sort t in descending order based Weightage
Add t to PS
Return PS
End



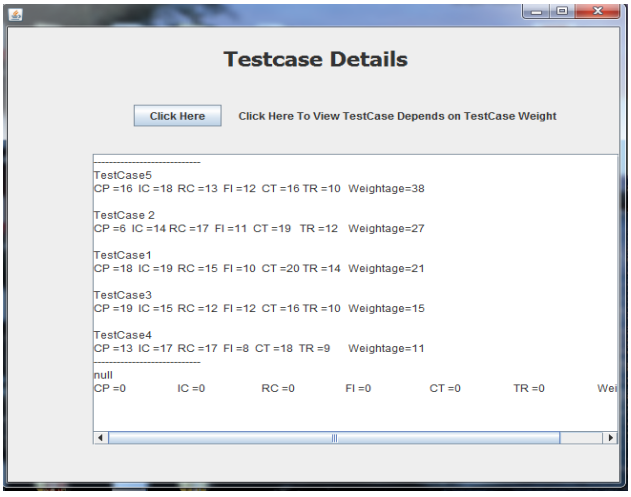
(a)



(b)



(c)



(d)

Fig. 2 The samples of (a).Requirement for entering account number (The field must be in integer), (b). the sample screen for withdrawal operation, (c). The fault occurs during the bank account creation for the same account number, (d). Final screen for proposed prioritization technique

4. EXPERIMENTAL RESULTS AND ANALYSIS

The test case prioritization system is proposed in this paper was implemented in the platform of java (JDK 1.6). Here we can use bank application system for regression testing and the results during the process are described as follows: We can create test cases for banking application to check their functionalities. Fig 1 shows that the initial screen obtained for regression testing. The user must enter the details which satisfy the certain constraints and data must be saved in the database regarding the operations of the banking applications. Test cases are generated for every wrong details entered by the user, if the requirements for the specific operations are not satisfied, adequate number of test cases is generated by our proposed system. After entering the account details for a particular user account is entered, account number must be unique i.e., the field should be in integer and this can be described in **Fig.2a.** During withdrawal operation, the requirement for account number should be an integer for the specific bank, and the test case is generated during this operation which can be described in **Fig.2b**. In **Fig.2c**, the field account number is already stored and it should be unique so here is the major fault occurred and the test case is generated and shown. The above figure describes the final output after regression testing. After performing the possible test conditions for each requirement in the banking application, test cases are generated. Based on the proposed approach we can prioritize the generated test cases using the factor values. We can sort the test cases based on test case weight age and the results are described in the **Fig. 2d.**

5. DISCUSSIONS

Here we can evaluate the effectiveness of the proposed prioritization technique by means of APFD metric and also by comparing the results with random ordered execution. The test suite has been developed for banking application project which consisting of 5 test cases and it covering a total of 5 faults. The regression test suite *T* contains 5 test cases with default ordering {T1, T2, T3, T4, and T5} and the number of faults occurs during the regression testing {F1, F2, F3, F4, and F5}. The test case results are shown in the **Table 1**.

Testcases/ Faults	T1	T2	T3	T4	T5
F1	x				
F2	x			x	x
F3	x			x	x
F4		x	x		x
F5			x	x	x
No.of faults	3	1	2	3	4

Table 1: fault detected by test suites in bank project

5.1 APFD Metric

The metric of Average Percentage of Fault Detected (APFD) is widely used for evaluating test case prioritization techniques. Let *T* be a test suite containing n test cases, *F* be a set of *m* faults revealed by *T*, and TF_i be the first test case index in ordering *T'*of T that reveals fault i. The following equation shows the APFD value for ordering *T'*

$$\text{APFD} = 1 - \frac{TF1 + TF2 + \ldots + TFm}{nm} + \left(\frac{1}{2n}\right) \qquad (4)$$

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

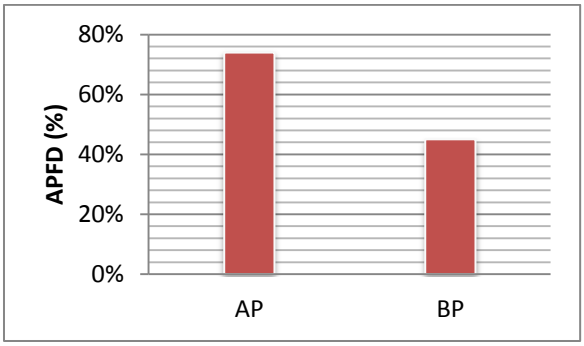


Fig. 3 APFD metric for test cases

test cases executed. In our paper, we can use the APFD metric for the performance based evaluation and the proposed test sequence is {T5, T2, T1,T3,T4}. Then the APFD metric after prioritization is APFD(T,P) is 0.74 and the APFD metric before prioritization is APFD(T,P) is 0.45 as per our above formula. **Fig. 3** Shows that the APFD metric comparison for both prioritized and non-prioritized test suite.

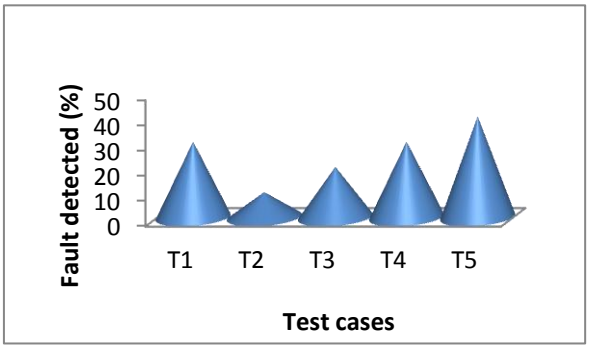


Fig. 4 Fault identified by each test case.

From the above figure shows that the test case 5 which detects more number of faults and it is shown in **Fig.4**. In the prioritized test suite total number of faults can be identified is more when compared with the random execution of test sequene and it can be shown in the **Fig.5**.

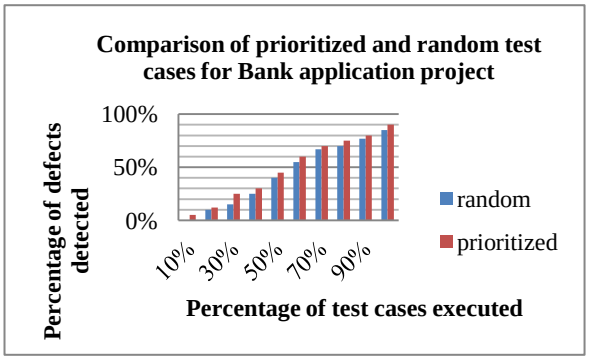


Fig. 5 TSFD is higher for prioritized test case reveals more defects.

Thus the prioritized test cases return better fault detection than the non – prioritized test cases and our proposed method of test case prioritization process will reduce the re-execution time of the project by prioritizing the most important test cases.

6. CONCLUSIONS

In this paper, we proposed 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 are tested for the regression test cases. The proposed prioritization algorithm is validated by using APFD metric. 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] Badhera, Usha; Purohit G.N. Biswas, Debarupa. 2012.Test Case Prioritization Algorithm Based Upon Modified Code Coverage In regression Testing.International Journal Of Software Engineering &Applications.Vol. 3 Issue 6, pp.29-34.

[2] BixinLi , Dong Qiu , Hareton Leung , Di Wang.2012. Automatic test case selection for regression testing of composite service based on extensible BPEL flow graph. Journal of Systems and Software. Vol:85 n.6, pp.1300-1324.

[3] B. Jiang, Z. Zhang, W.K Chan, T.H Tse, Adaptive random test case prioritization, in: Proceedings of the 24th IEEE/ACM International Conference on Automated Software Engineering (ASE 2009), IEEE Computer Soceity press, Los Alamitos, CA, 2009, pp.233-244.

[4] Dr. ArvinderKaur and ShubhraGoyal. 2011. A Genetic Algorithm for Fault based Regression Test Case Prioritization. International Journal of Computer Applications. Vol: 32(8).pp:30-37.

[5] Hong Mei, Dan Hao, LingmingZhang,Lu Zhang, Ji Zhou, and Gregg Rothermel. 2012.A Static Approach to Prioritizing JUnit Test Cases.IEEE Transactions On Software Engineering, Vol. 38, No. 6.

[6] H. Do, S. Mirarab, L. Tahvildari, and G. Rothermel.2010. The Effects of Time Constraints on Test Case Prioritization:ASeries of Controlled Experiments. IEEE Trans. Software Eng.Vol:36. no. 5.pp:593-617.

[7] H. Park, H. Ryu, J. Baik.2008. 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. pp. 39–46.

[8] Mohamed A Shameem and N Kanagavalli.2013. Dependency Detection for Regression Testing using Test Case Prioritization Techniques. International Journal of Computer ApplicationsVol 65(14): pp:20-25.

[9] M. Yoon, E. Lee, M. Song and B. Choi.2012. A Test Case Prioritization through Correlation of Requirement and Risk. Journal of Software Engineering and Applications. Vol. 5 No. 10. pp. 823-835. doi: 10.4236/jsea.2012.510095.

[10] R. Abreu, P. Zoeteweij, A.J.C. van Gemund.2009. Spectrum-based multiple faultlocalization, in: Proceedings of the 24th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 88–99.

[11] R. Bryce, S. Sampath, and A. Memon.2011. Developing a Single Model and Test Prioritization Strategies for Event-Driven Software. IEEE Trans. Software Eng. Vol. 37. no. 1. pp. 48-64.

[12] R. Krishnamoorthi and S. A. Mary.2009.Factor Oriented Requirement Coverage Based System Test Case Prioritization of New and Regression Test Cases. Information and Software Technology. Vol. 51.No. 4. pp. 799-808.

[13] S. Raju and G.V. Uma.2012. An Efficient method to Achieve Effective Test Case Prioritization in Regression Testing using Prioritization Factors. Asian Journal of Information Technology. Vol:11.issue:5.pp:169-180. DOI: 10.3923/ajit.2012.169.180