EFFECTIVENESS OF TEST CASE PRIORITIZATION TECHNIQUES BASED ON REGRESSION TESTING

Thillaikarasi Muthusamy[1] and Dr. Seetharaman.K[2]

[1]Department of computer science,Annamalai University,Annamalai Nagar, Tamilnadu,India
[2] Department of computer science,Annamalai University,Annamalai Nagar, Tamilnadu,India

ABSTRACT

Regression testing concentrates on finding defects after a major code change has occurred. Specifically, it exposes software regressions or old bugs that have reappeared. It is an expensive testing process that has been estimated to account for almost half of the cost of software maintenance. To improve the regression testing process, test case prioritization techniques organizes the execution level of test cases. Further, it gives an improved rate of fault identification, when test suites cannot run to completion.

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 improperly. Hence, Regression Testing becomes essential in software testing process. this leads to regression testing in which all the tests in the accessible programmes or suite should be re-executed. Thus incurring excess cost, time and resources. Test case prioritization is an important technique adopted in regression testing. Prioritize the test cases depending on business impact, importance & frequently used functionalities. Selection of test cases based on priority will significantly reduce the regression test suite. Here we suggest a new approach for test case prioritization for prior fault detection in the regression testing process.

Here, we have projected a new approach to test case prioritization for quick fault detection based on practical influences. We have implemented the proposed technique in a banking application project and effectiveness is calculated by using APFD metric.

DOI : 10.5121/ijsea.2014.5608

2. TECHNIQUES REVISITED

This segment narrates about the test case prioritization techniques to be used in our empirical study are as follows:

Test case prioritization is an important regression testing technique, which approaches typically and restructure existing test cases for regression testing accordingly to achieve targets.

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 order such that fewer lines of code need to be executed. Thus faster code coverage is attained which will 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] projected an ART-based prioritization method which uses the algorithm and accepts the test suite as input, produces the output in prioritized order. The basic idea is about 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 also to select a test case from the candidate set. Calculation of distance is determined by code coverage data. Then we find a candidate test case which is related with the distance test cases that has been prioritized earlier.

Dr. ArvinderKaur and ShubhraGoyal [3] developed a new genetic algorithm and prioritize regression test suite within a time constrained environment on the basis of entire 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 is widely used in java programs under the JUnit framework. A new approach called JUPTA which operates in the absence of coverage information and analyzes the static call graphs of JUnit test cases. Further, it estimates the ability of each test case to achieve code coverage and schedules the test cases in an order based on those estimates.

H.Do et al. [5] presented the importance of time constraints on test case prioritization and discovered that constraints which alters the performance of technique. Further, conducted three set of experiments which reveals the time constraints. The outcome show that the time constraint factor play a significant role in determining the cost effectiveness and cost benefit trade-offs among the techniques. Next experiment reproduces the first experiment, calculating several threats to validate numbers of faults present. Third experiment manipulates the number of faults present in programs to examine the effect of inaccuracy on prioritization and exhibits the relative cost-effectiveness of prioritization techniques.

Park et al. [6] introduced a cost awareness model for the test case prioritization and fault severities which revealed in the previous test execution. As well as it does not significantly change form one outcome to another. Mohamed A Shameem et al. (2013) presented a metric for

assessing the rate of fault detection. This algorithm identifies the faults in prior and the effectiveness of prioritized test cases are compared with the non prioritized cases by Average Percentage Of Fault Detection (APFD).

M. Yoon et al. [7] proposed a method to prioritize new test cases by estimating the requirements of risk exposure value and also analyzing risk objects. Further it calculates the relevant test cases and thereby determining the test case priority through the evaluated values. Moreover, we demonstrate the effectiveness of our technique through empirical studies in terms of Average Percentage Of Fault Detected (APFD) and fault severity.

R. Abreu et al. [8] projected a Spectrum-based multiple fault localization method to find out the fault location apparently. R. Bryce et al. (2011) suggested a model which describes prioritization criteria for GUI and web applications in an event driven software. The ultimate purpose is to evolve the model and to develop a unified theory about testing of EDS.

R. Krishnamoorthi and S. A. Mary [9] presented a model that 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 examined in three phases with student projects and two sets of industrial projects. Here results were found to improve the rate of severe fault detection

S. Raju and G.V. Uma [10] initiated a cluster-based test case prioritization technique. Here, the test cases are clustered based on their dynamic runtime behavior. Significantly researchers reduced the required number of pair-wise comparisons. Researchers presented a value-driven approach to system-level test case prioritization which prioritizes the requirements for test. Here, prioritization of test cases is based on four factors: rate of fault detection, requirements volatility, fault impact and implementation complexity.

The rest of this paper is organized as follows. Section three discusses about the proposed work. Section four discusses about the experimental results and analysis. Section five discusses about the test cases to be prioritized. Finally, section six consists of conclusion. References are given in last section.

3. PROPOSED WORK

This section, briefly discusses about the prioritization factors.

3.1 Prioritization Weight Factors

It deals with, computation of 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 which is essential for prioritizing the test cases since they are used in the prioritization algorithm. Weights are assigned to each test case in the software testing according to the factors. Then, test cases are prioritized based on the weights assigned.

3.1.1 Customer-Allotted Priority (CP)

It determines the requirements of the customer and the value are assigned by the customers. The values vary from 1 to 20, where 20 are used to identify the highest customer priority. So, improving customer's fulfillment imposes, initial testing of the highest priority needs of the customer. Greater effort should be taken in identifying faults and their impacts 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 complexity expected by the development team to implement the requirements. First every necessity is evaluated by assigning a value from 1 to 20. Based on the implementation complexity, the higher complexity is implied by a larger value. Large number of faults that occurs in a requirement 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 which indicates that the requirement is changed as many times during the development cycle with respect to its origin. The volatility values for all the needs are expressed on a 20-point scale where the need is altered more than 20 times. The number of changes for any requirement 'i' is divided to the highest number of changes which in turn yields the change in requirement *Ri* where the requirement is 'i'. If the ith requirement is changed M times and N is the maximum number of requirements, then the requirement change *Ri* can be calculated as follows:

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

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

3.1.4 Fault Impact of Requirements (FI)

It allows the development team to differentiate the requirements 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 since system evolves to several versions. The number of in-house failures and field failures determine the fault impact of requirements. It is a measure for released product. It is proved that field failures are more likely to be fault prone modules than modules that are not fault prone.

3.1.5 Completeness (CT)

This part indicates requirement based function to be executed, the rate of success, the limitations and any limitation which manipulate the expected solution. (boundary constraints). The consumer assigns value from 1 to 20. When the condition is selected for reuse by scrutinizing the completeness of each requirement into consideration, customer satisfaction can be enhanced.

3.1.6 Traceability (TR)

Relation between requirement and assessment can be calibrated by means of Traceability. 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 is defined as ability to monitor life of requirement in either ways i.e. from the inception through construction, specification, subsequent execution and usage through 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, the standard of software can be improved by opting the traceability into consideration is chosen for subsequent usage.

3.2 Proposed Prioritization Algorithm:

Values for all the 6 factors are assigned for each test case and analyzed continuously during the software development process. We can compute weighted prioritization value (WPV) for each test case i shown in Eqn(2)

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

Where, WPV is weight prioritization value for each test case are 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 to compute 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

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

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. **Figure 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.

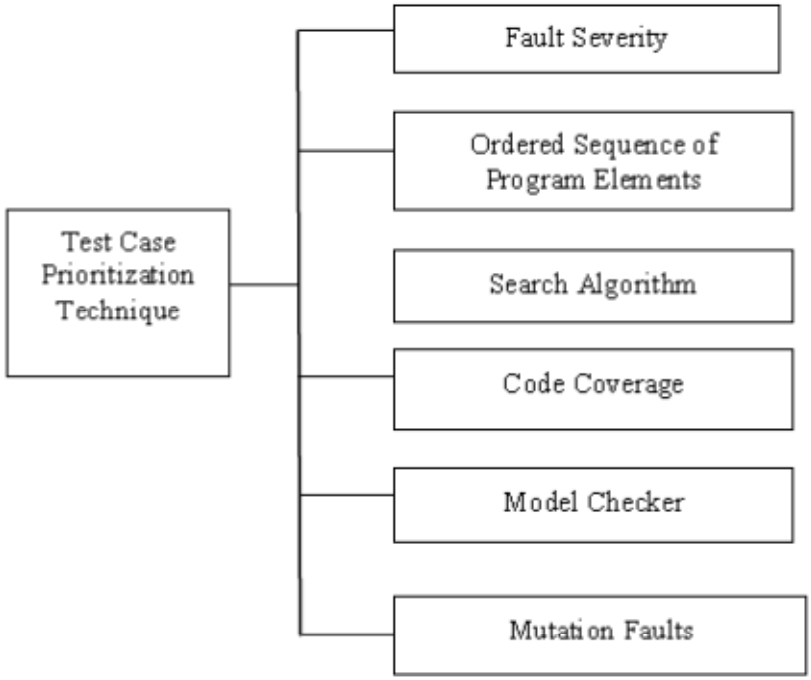


Fig 2. Test case prioritization techniques

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

Step 1. Begin
Step 2. Set T' empty
Step 3. For each t ∈ *T* do
Step 4. Calculate the 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
Step9.End

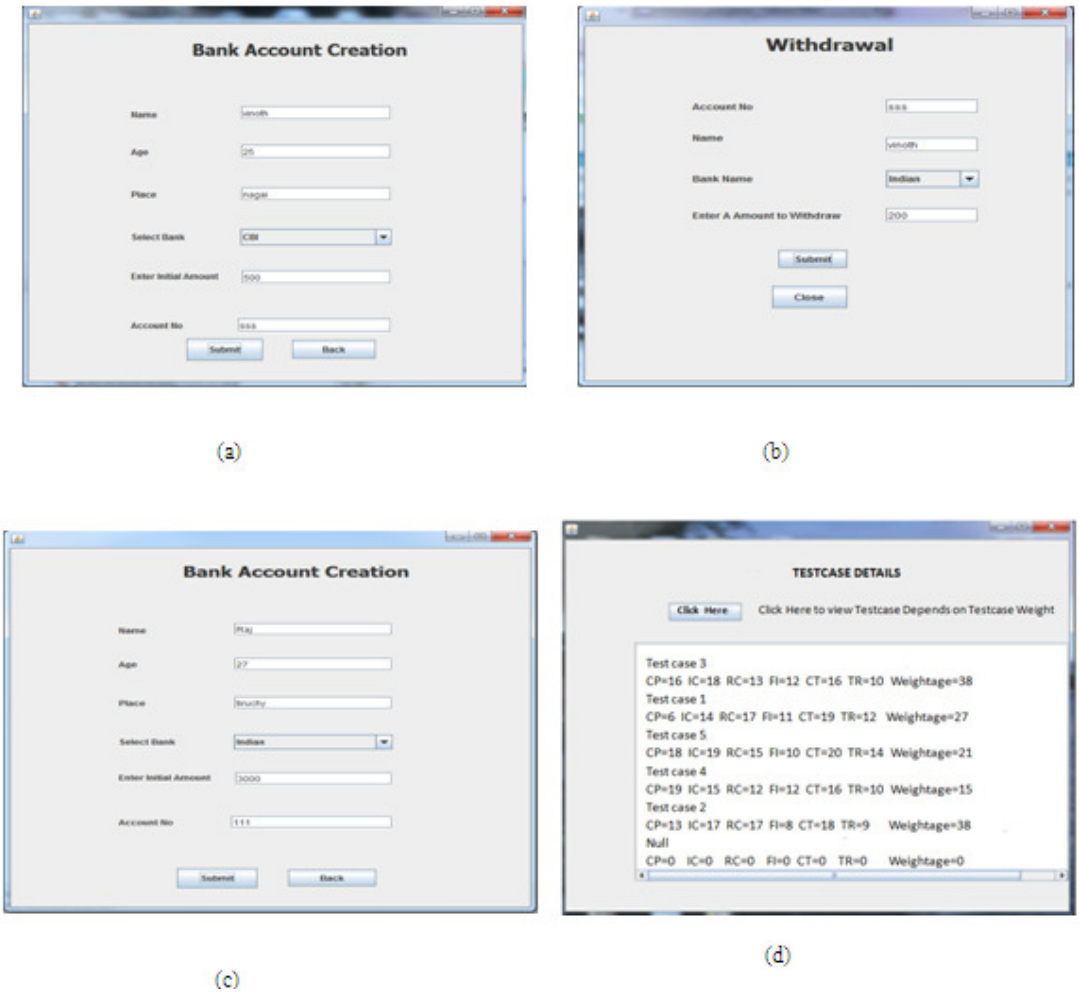


Figure 2. The samples of (a).Requirement for entering account number (The field must be in integer), (b). the sample screen for withdrawal operation, (c). The fault occurs during the bank account creation for the same account number, (d). Final screen for proposed prioritization technique

4. EXPERIMENTAL RESULTS AND ANALYSIS

The test case prioritization system is proposed in this paper which 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 2 shows that the initial screen obtained for regression testing. When user enters the details it satisfies certain constraints and data must be saved in the database with regard to 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, sufficient number of test cases are generated by our proposed system.

After entering the account details for a particular user account, the account number must be unique i.e., the field should be in integer and this can be described in **Figure 2a.** During withdrawal operation, the requirement for account number should be an integer for a definite bank and the test case is generated during this operation can be described in **Figure 2b**. In **Figure 2c**, the field account number is already stored and it should be unique so that the major fault is occurred and the test case are generated and shown. The above figure describes the final output after regression testing. After executing the possible test conditions for each requirement in the banking application, test case are generated. In view of the above we can prioritize the generated test cases using the factor values. Then, we can sort the test cases based on test case weightage and the results are exhibited in the **Figure 2d.**

5. DISCUSSIONS

Here we can evaluate the effectiveness of the proposed prioritization technique by means of APFD metric and the results are compared with random ordered execution. The test suite has been developed for banking application project which consists of 5 test cases and it covers 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**.

Table 1. Fault Detected By Test Suites In Bank Project

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

5.1 APFD Metric

The test case prioritization techniques is evaluated by metric of Average Percentage of Fault Detected (APFD*)*. 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* 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 that it produces statistically significant results. The APFD measures that the average number of faults are detected in a given test suite. The

APFD values ranges from 0 to 100 and percentage of fault are detected by plotting the area under the curve towards the percentage of test case executed.

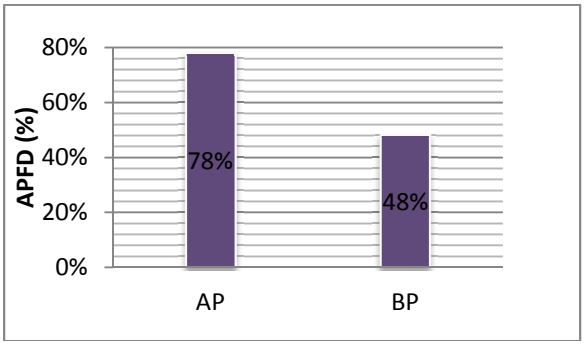
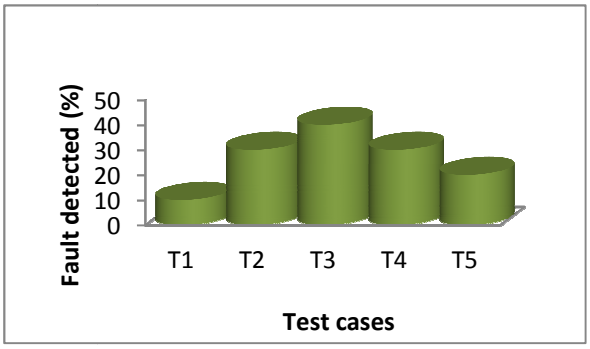


Figure 3. APFD metric for test cases

In this paper, APFD metric is used and the proposed test sequence is {T3, T1, T5, T4, T2}. 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 the above formula. **Figure 3** Shows that the APFD metric compares both prioritized and non-prioritized test suite.



Figrure 4. Fault identified by each test case.

The above figure shows that the test case 5 detects more number of faults and it is shown in **Figure 4**. In the prioritized test suite, more number of faults can be identified when compared with the random execution of test sequence and the same is shown in the **Figure 5**.

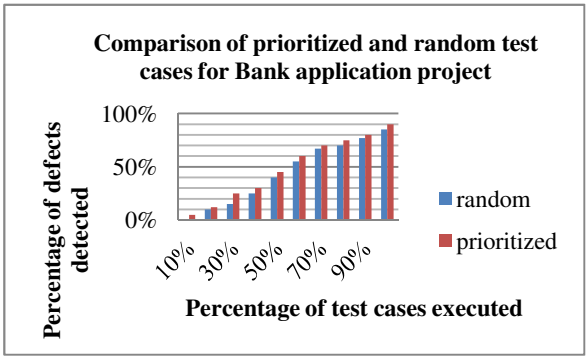


Figure 5. TSFD is higher for prioritized test case which reveals more defects.

Thus the prioritized test cases provides better fault detection than the non – prioritized test cases. Further test case prioritization technique will reduce the processing time of the project by prioritizing the most important test cases.

6. CONCLUSIONS

Our algorithm is based on analysis of the percentage of test cases performed to find the faults and on APFD metric's results. Abiding by the percentage of executing test cases in earlier fault detection is important as sometimes regression testing ends without executing all test instances. Outcomes demonstrate that our algorithms can also achieve better execution in this event. For instance, in the first project if only 75% test cases could be melt down due to resource constraint, random strategy could find more or less 66% faults; while our proposed algorithm detects about 88% faults. In a second project if we consume 30% test cases to accomplish; then random strategy could find more or less 27% faults; while our proposed algorithm detects about 40% faults. This shows clear evidence that our proposed algorithm is a lot better in earlier fault detection than random technique. The graphical representation of these outcomes is presented at a lower place. We had also validated our results with the aid of standard APFD metric. We can discover the improved fault detection rate in earlier testing stage through our algorithm which is the proof of our algorithm; as more efficient and beneficial in earlier fault detection goal; whereas gradual improvement in APFD results are obtained by random strategy later in testing stage.

.

REFERENCES

[1] 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.

[2] 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.

[3] 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.

[4] 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.
[5] 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.
[6] 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
[7] R. Ramler, S. Biffl and P. Grunbacher "Value-Based Management of Software Testing",Book Chapter.
[8] B. Boehm and L. Huang, "Value-Based Software Engineering: A Case Study," IEEEComputer, vol. 36, pp. 33-41, Mar 2003.
[9] L. Zhang, S. S. Hou, C. Guo, T. Xie and H. Mei "Time Aware Test- Case Prioritization using Integer Linear Programming", ISSTA'09 , Chicago, Illinois,USA, Jul 2009
[10] Z. Li, M. Harman and R. M. Hierons "Search Algorithms for Regression Test Case Prioritization",IEEE Trans. on Software Engineering, vol. 33, no. 4, Apr 2007
[11] G. Rothermel, R. Untch, C. Chu and M. Harrold, "Test Case Prioritization: An Empirical Study" Int. Conf. on SoftwareMaintenance, Oxford, UK, pp. 179 - 188,Sep 1999.
[12] K. R. Walcott and M. L. Soffa "Time Aware Test Suite Prioritization", ISSTA'06,Portland, Maine, USA, Jul 2006
[13] M. Fayoumi, P. Mahanti and S. Banerjee: OptiTest: "Optimizing Test Case Using Hybrid Intelligence" World Congress on Engineering 2007.
[14] S. Mirarab and L.Tahvildari "An Empirical Study on Bayesian Network-based Approach for Test Case Prioritization" Int. Conf. on Software Testing, Verification and Validation 2008.
[15] K. H. S Hla, Y. Choi and J. S. Park "Applying Particle Swarm Optimization to Prioritizing Test Cases for Embedded Real Time Software Retesting", 8th IEEE Int. Conf.on Computer and on Information Technology Workshops 2008

AUTHORS

M.Thillaikarasi, working as a Assistant Professor in Department of Computer Science and Engineering, Annamalai University, Chidambaram, TamilNadu. She has finished B.E,M.E and pursuing Ph.D in the field of software engineering from Annamalai University She has teaching experience of 10 years. She has published 6 International
Journals.

Dr.K.Seetharaman, working as a Associate Professor in DDE – Engineering Wing(computer), Annamalai University, Chidambaram, TamilNadu. He has finished M.Sc.,PGDCA.,M.S.,Ph.D., He has teaching experience of 10 years and Research experience of 8 Years. He has published 18 International journals and 13 National journals.