Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 79 (2016) 152 – 160

Procedia Computer Science

7th International Conference on Communication, Computing and Virtualization 2016

Optimized Regression Test using Test Case Prioritization

Ahlam Ansari[a],Anam Khan[b],Alisha Khan[c],Konain Mukadam[d]

[a,b,c,d]*M.H. Saboo Siddik College of Engineering,8,Saboo Siddik Polytechnic Road,Byculla,Mumbai-400008,India.*
[a]*ahlam.shakeel@mhssce.ac.in* ,[b]*anam.khan@mhssce.ac.in,* [c]*alisha.khan@mhssce.ac.in,* [d]*konain.mukadam@mhssce.ac.in*

Abstract

Regression testing ensures that changes made in the fixes or any enhancement changes do not impact the previously working functionality. Whenever software is modified, a set of test cases are run to assure that changes don't affect the other parts of the software. Hence all existing test cases need to be tested as well as new test cases need to be created. It is nonviable to re-execute every test case for a given software, because if there are more number of test cases to be tested, the more effort and time is required. This problem can be solved by prioritizing test cases. Test case prioritization techniques reorder the priority of a test case in an attempt to ensure that maximum faults are uncovered by the high prioritized test cases. In this paper we propose an optimized test case prioritization technique using Ant Colony Optimization (ACO) to reduce the cost, effort and time taken to perform regression testing and also uncover maximum faults.

*Keywords:*Regression testing;Test case prioritization;Ant colony optimization

1. Introduction

When the software is designed then the software development life cycle is used to develop the complete software. For the software development, different phases are followed. After the designing phase testing phase is used to test the whole design code. There are different types of testing that are used to test the code. The regression testing is one of the types of the testing that is used to test the software code[1].

Regression testing is a type of testing in which previously executed test cases are re-run so as to verify that existing functionalities work fine. This testing is done to make sure that the process of fixing existing bugs doesn't result in new bugs, affecting, previously flawless, functions. Regression Testing is required when there is

* Corresponding author. Tel.: +0-000-000-0000 ; fax: +0-000-000-0000 .
E-mail address: ahlam.shakeel@mhssce.ac.in

- Change in requirements and code is modified according to requirement
- New features are added to the software
- Defect fixing Performance issue fix[2].

There are various techniques for regression testing. The different regression testing techniques are listed below[3]:

- Retest All
- Regression Test Selection
- Test Case Prioritization
- Hybrid Approach.

Fig 1 shows the various Regression Testing Techniques

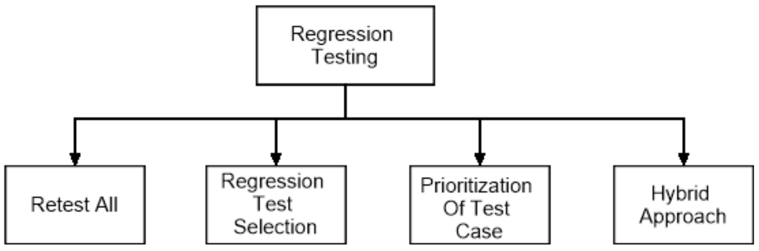


Fig 1. Regression Testing Techniques

- Retest All: One of the methods of performing regression testing is to retest all the test cases i.e. all the tests in the existing test suite should be re-executed. Running all test cases in an existing test suite, however, can consume a lot of time.
- Regression Test Selection: Instead of re-executing the entire test suite, it is better to select part of test suite to be run. Test cases selected can be categorized as I) Reusable Test Cases II) Obsolete Test Cases. Reusable Test cases can be used in succeeding regression cycles. Obsolete Test Cases can't be used in succeeding cycles.
- Prioritization of Test Cases: This way deals with prioritizing the test cases such that test cases with higher priority, according to some criterion are executed earlier than those with lower priority to meet some performance goal. So test case prioritization technique does not discard test cases, they can avoid the drawback of test case minimization technique.
- Hybrid Approach: The fourth regression testing is the Hybrid Approach of both Regression Test Selection and Test Case Prioritization.

Various prioritization criteria may be applied to the regression test suite with the objective of meeting a certain criterion. Test cases can be prioritized in terms of random, optimal, statement coverage total or additional, branch coverage total or additional, failure rates, or total fault exposing potential of the test cases[4]. Many techniques have been used for prioritization according to one or more of the chosen criteria. In this paper we make use of ACO to prioritize the given test cases. ACO is a process based on real life of ants, precisely on their food searching process.

2. Regression Testing

The maintenance phase of a software product needs to go through the inevitable regression testing process. It is required to retest the existing test suite whenever any addition, deletion or modifications are made to the software. Re-running the test cases from the existing test suite to build confidence in the correctness of the modified software is referred to as regression testing[5].

Regression testing is a type of testing that guarantees that any enhancement or defect fixes in the software or its environment do not impact the previously working functionalities of the software. While testing the working of a specific function it might introduce new defects in other functions.

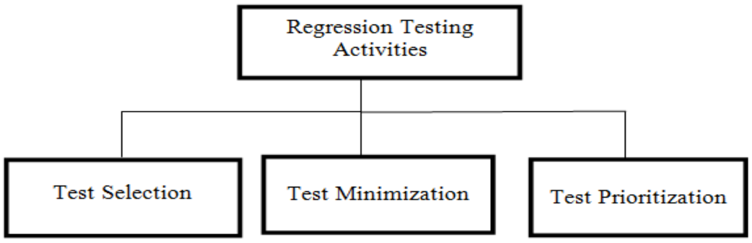


Fig. 2. Various activities that are carried out during regression testing.

Regression testing focuses on the criteria that the changes done in a particular function should not introduce any defects in the previous flawlessly working functions. Various regression testing techniques can be seen in Fig. 2.

3. Test Case Prioritization

Test case prioritization schedules test case according to some criterion. The objective of this technique is to enhance the possibility that if the test cases are prioritized in some order, they will meet the specified goals within stipulated time and cost than they would if they were not prioritized.

Test case prioritization can address a wide variety of objectives, as given below:

- Software developers/testers intend to increase the rate of fault detection.
- Detecting the high-risk faults earlier in testing life cycle.
- To increase the possibility of regression errors related to specific code changes very early in testing process
- To enhance the coverage of coverable code at a faster rate.
- To make a system more reliable[6].

3.1. Test case prioritization techniques

Test case prioritization technique involves the selection of test cases that uncover maximum faults in the components of the software and assigning high priorities to test cases that have minimum execution time and are used maximum times. Execution of these prioritized test cases proves to be sufficient for testing the functionalities of components of a software in a lesser time. Test case prioritization thus helps in reducing the time and efforts thereby reducing the cost of testing.

The nine different test case prioritization techniques are no prioritization, random prioritization, optimal prioritization, total branch coverage prioritization, total fault-exposing-potential (FEP) prioritization, additional fault-exposing-potential (FEP) prioritization, total statement coverage prioritization, and additional statement coverage prioritization[7].

3.2. Algorithms for test case prioritization

There are many algorithms for test case prioritization which are being developed and unfolded by various researchers in the field. They are Greedy Algorithm, Additional Greedy Algorithm, Optimal Algorithm, Hill Climbing Algorithm, Genetic Algorithm, PORT version1.1, Ant Colony optimization[3].

4. Ant Colony Optimization

Ant Colony Optimization is an optimal path searching technique that is based on the natural behaviour of ants on a food hunt. During food hunt all the ants follow a certain path and each ant leaves a chemical substance behind it which is known as “pheromone”. The ants following the same path are able to trace the path by smelling the odour of pheromone that was left behind[8].

The shortest path will be discovered through teamwork and pheromone evaporation process. As seen in Fig. 3 a. Random path followed by ants [8]there are two possible paths from the ant nest to the food source. Assume in the beginning of food search, ant choose their path randomly depositing pheromone on each path. But in the return trip, the ants will choose the path which has more residual pheromone. Since the pheromone evaporates according to the length of paths, thus each path will have different amount of residual pheromone.

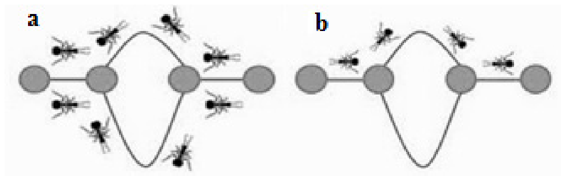


Fig 3. (a) Random path followed by ants (b) Shorter path followed by ants

The longer the path, will thus have more pheromone evaporation and less residual pheromones. The other follower ants will choose the shorter path according to the amount of residual pheromones as seen in Fig. 3 b shorter path followed by ants[8].

5. Optimized Test Case Evaluation Metric

It is a must to assess effectiveness of the sequence/ordering of the test case using some metric. The proposed technique is evaluated by metric of Average Percentage of Fault Detected (APFD).

5.1 Need for optimization of test cases

Re-running all existing test cases is often costly and sometimes even infeasible due to time and resource constraints. Thus arises the need to optimize test case to maximize coverage and minimize redundancy with minimum time and resources utilised. An optimized test case will not only help provide a baseline assessment of the current version, but some of those tests will also be reused on the next release or version of the software. Such optimized test case will enable testers to rapidly re-verify the changes and quickly respond to new function requirements.

5.2 Test case Evaluation attributes

Following are the attributes of test case evaluation:

- Size: Size of the set of test cases (test suites) corresponds to the number of test cases present in it.

$$Size = The\ count\ of\ Test\ Case \quad (1)$$

- Effort: Effort is the ratio of size of test suite and number of test cases to be executed in a man day. It is calculated as follows:

$$Effort = Size \div No. of\ Test\ case\ to\ be\ executed\ in\ a\ Man\ day\ (Man\ day/Person) \qquad (2)$$

- Time: Time required to test a set of test cases depends on the size of test suites and time taken per person to run and test the test cases. It is calculated as:

$$Time = Size \times Time\ required\ per\ person\ (Hour) \qquad (3)$$

- Cost: Cost for executing a set of test cases depends upon the effort and cost of per man day. Cost of execution is calculated as:

$$Cost = Effort \times Cost\ of\ per\ man\ day\ (\$) \qquad (4)$$

5.3 APFD matrix

To quantify the goal of increasing a subset of the test suite's rate of fault detection, we use a metric called APFD developed by Elbaum et al.[9] that measures the rate of fault detection per percentage of test suite execution. The APFD is calculated by taking the weighted average of the percentage of faults detected during the execution of the test suite. Higher APFD values imply faster (better) fault detection rates. APFD can be calculated as follows:

$$APFD = 1 - \left\{\frac{Tf1 + Tf2 + \cdots Tfm}{mn}\right\} + \frac{1}{2n} \qquad (5)$$

Where,
T → The test suite under evaluation
m → the number of faults contained in the program under test P
n →The total number of test cases
Tfi → The position of the first test in T that exposes fault i.

So as the formula for APFD shows that calculating APFD is only possible when prior knowledge of faults is available. The APFD metric relies on two assumptions: (1) all faults have equal costs. (2) All test cases have equal costs. These assumptions are manifested in the fact that the metric plots the percentage of faults detected against the percentage of the test suite run.

6. Proposed Technique

The paper mainly focuses at optimizing the efficiency of regression testing by sorting the list of test cases, required for a successful regression test of the software. This is done by taking input of test case matrices and sorting it in accordance to certain criterion. Thus the efficiency of the test will be optimized.

Initially a test case is chosen that covers maximum faults. Since the aim is to uncover all faults, thus it is checked if all faults are covered by it or not. If not, then choose the next test case that covers the remaining faults and repeat this until all faults have been covered. Once all faults are covered, calculate the total number of faults covered by each test case which is stored in total fault test case matrix. This procedure will lead to many combinations of test case called paths that cover all faults.

Many such paths are explored while iterating and the best path from all paths explored is selected. The pheromone value is updated on the best path selected. The selection of best path is based on minimum execution time, maximum probability of usage and highest pheromone value.

This process of prioritizing using Ant Colony Optimization is further explained with help of an algorithm.

6.1 Proposed Algorithm

Inputs: Fault Test case matrix, Execution time test case matrix, Usage matrix, Pheromone matrix.
Output: Prioritized list of test case.

Step 1: Input to the system the following:
a) Fault Test case matrix
b) Execution time test case matrix
c) Usage matrix

d) Pheromone matrix

Step 2: Initially the pheromone value for each test case is calculated as:

$$\text{Pheromone Value} = \text{Faults Covered} \div \text{Execution Time} \qquad (6)$$

Step 3: Iterate for test cases in the test pool

Step 4: Select a test case that covers maximum fault. Check if all faults are covered by it. If not, select other test cases that cover the remaining faults.

Step 5: Terminate the iteration when all faults are covered.

Step 6: Calculate the total faults covered by test cases for all requirements and store it in the Total fault test case matrix.

Step 7: Explore all the paths in each iteration.

Step 8: For each path calculate:

a) Average execution time of test case

b) Average probability of usage of test case

Step 9: Compare the paths based on average execution time of test case, average probability of usage of test case and its pheromone value.

Step 10: Select the best path.

Step 11: Update the pheromone value of the test case for the best path chosen.

7. Results

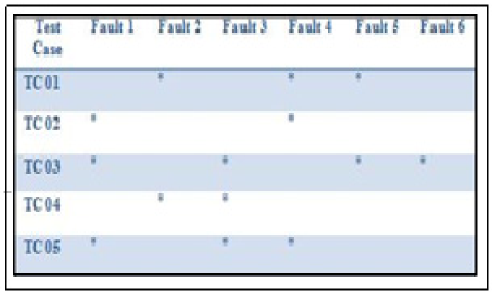
The results are bifurcated into output section which consists of various screenshots of inputs and output obtained by applying the proposed algorithm. The next section is comparison results that compares output of proposed algorithm with manual technique on different criteria.

7.1. Output

The outputs are shown below. Five test cases along with their faults uncovered and execution time is taken as input. Also the pheromone value for each test case is taken as input.

The Fault test case matrix is shown in Fig. 5 a. The faults uncovered by each test case for a particular requirement are checked and stored in this matrix. The faults which are uncovered by a certain Test case are marked as ‘*’.

a



Test Case	Fault 1	Fault 2	Fault 3	Fault 4	Fault 5	Fault 6
TC 01		*		*	*	
TC 02	*			*		
TC 03	*		*		*	*
TC 04		*	*			
TC 05	*		*	*		

b

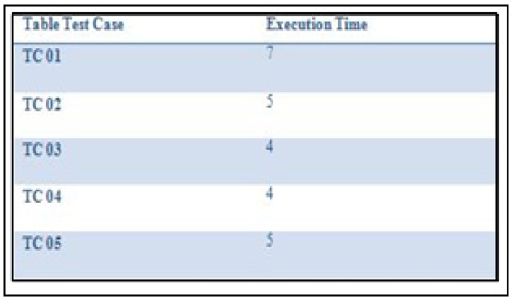
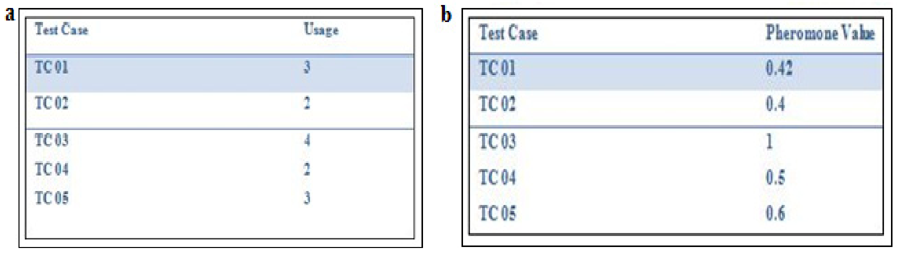


Table Test Case	Execution Time
TC 01	7
TC 02	5
TC 03	4
TC 04	4
TC 05	5

Fig. 4. (a) Fault test case matrix (b) Execution time test case matrix

From Fig. 4 a it can be said that TC 03 covers maximum faults and the remaining faults are covered by TC 01. The execution time of a test case is calculated based on length of the script of the Test case. Longer the script of Test case more will be its execution time. The execution time calculated for each Test case is stored in the Execution time test case matrix. The Execution time test case matrix is shown in Fig. 4 b. Usage of a Test case is calculated based on the previous usage of a Test case. Usage matrix displays the number of times a particular Test case has been in use.

The Usage matrix is shown in Fig. 5a.



a

Test Case	Usage
TC 01	3
TC 02	2
TC 03	4
TC 04	2
TC 05	3

b

Test Case	Pheromone Value
TC 01	0.42
TC 02	0.4
TC 03	1
TC 04	0.5
TC 05	0.6

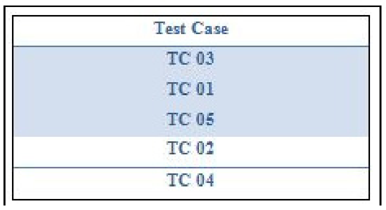
Fig. 5 a, b

From Fig. 5 a it can be concluded that Test cases TC 03, TC 01 and TC 05 are used maximum times.

The pheromone values for each Test case are stored in the Pheromone matrix. It displays the test cases and their respective pheromone values. The initial pheromone value for Test case is calculated as per Equation 6.

According to the best path selected the pheromone values are updated. Selection of best path is done based on average execution time of test case, average probability of usage of test case and its pheromone value. The pheromone matrix is shown in Fig. 5 b.

The test cases are prioritized based on their fault coverage, execution time, and usage and pheromone value. The test case that uncovers maximum faults is selected. If it does not cover all the faults, then other test case(s) which uncovers remaining faults is/are selected. The prioritized test cases are shown in Fig. 6.



Test Case
TC 03
TC 01
TC 05
TC 02
TC 04

Fig. 6. Prioritized test case matrix

From the results it can be seen that TC 03 and TC 01 cover all the faults of the software. Thus out of five test cases only two of them cover all the faults within minimum execution time hence the best path is selected with TC 03 and TC 01.

7.2. Comparison Results

The comparative results between proposed technique and manual testing using different criteria are shown below based on the attributes of comparison i.e. Size, Effort, Time, and Cost. The Proposed Technique optimizes test case based on multiple objectives and thereby reduces the effort, cost and time required in testing which is more in manual testing.

The Fig. 7 a shows the comparison between proposed technique and manual technique based on size of representative set generated.

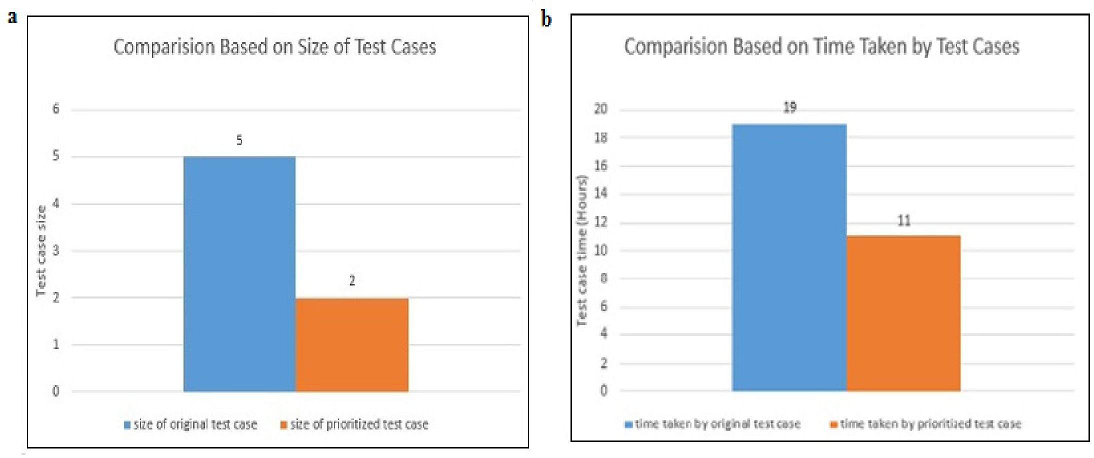


Fig. 7. (a) Comparison based on size of test cases (b) Comparison based on time taken by test cases.

The Fig. 7 b shows the comparison between proposed technique and manual technique based on time taken by test cases.
The Fig. 8 a shows the comparison between proposed technique and manual technique based on effort taken by test cases.

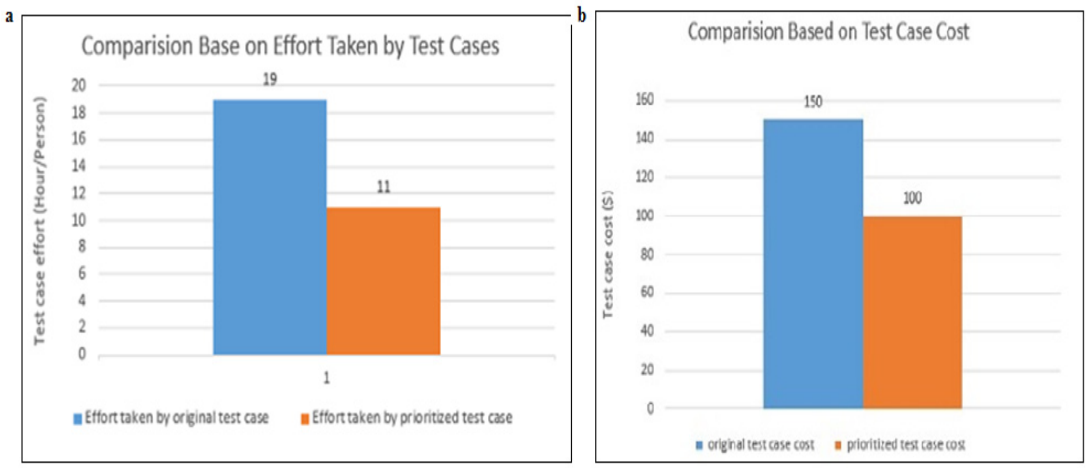


Fig. 8. (a) Comparison based on effort taken by test cases (b) Comparison based on cost of test cases.

The Fig. 8 b shows the comparison between proposed technique and manual technique based on cost of the test cases.

The effectiveness of the proposed algorithm will be measured by the rate of faults detected. The APFD metric is used to calculate the level of effectiveness. To calculate APFD, we consider the fault test case matrix with 6 faults and 5 test cases as shown in Fig. 4 a. Here comparison among the results of prioritized and non- prioritized test case is done based on the results of the APFD metric. The prioritized order according to the proposed technique is:
T3 T1 T2 T4 T5.
Applying APFD w.r.t prioritized test cases:

APFD= 1-{(1+2+1+2+1+1+1) / (5*6)}+{1/(2*5)} = 0.6333
Applying APFD w.r.t non-prioritized test cases:
APFD=1-{(2+1+3+1+1+3) / (5*6)}+{1/(2*5)} = 0.5333

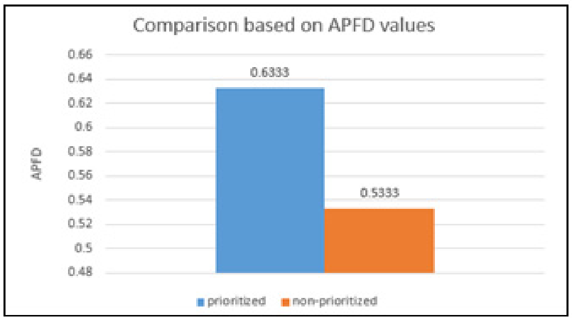


Fig. 9. Comparison based on APFD values.

Thus the prioritized test cases yield better fault detection than the non – prioritized test cases as shown in the Fig. 9.

8. Conclusion

The proposed technique for test case prioritization in regression testing will automatically optimize list of test cases for a specified function to make the regression testing as efficient as possible. Test cases are prioritized and optimized using ant colony optimization technique. Prioritizing test cases will minimize the time, effort and cost of testing and uncover maximum faults in the software. Deployment of this algorithm would undeniably prove to be great reduction in the testers' effort and time complexity.

Acknowledgement

Our thanks to M.H. Saboo Siddik College of Engineering, Department of Computer Engineering, for giving us the initiative to do constructive work. We also thank anonymous reviewers for their constructive suggestions.

References

1. Neha Sethi, Shaveta Rani and Paramjeet Singh, "Ants Optimization for Minimal Test Case Selection and Prioritization as to Reduce the Cost of Regression Testing", *International Journal of Computer Applications*, 2014.
2. Yogesh Bhardwaj and Dr. Manju Kaushik, "A Review Paper on Effort Estimation and Model Based Regression Testing with SOA", *International Journal of Computer Science and Mobile Computing*, 2014.
3. Gaurav Duggal, Mrs. Bharti Suri, "Understanding Regression Testing Techniques", Guru Gobind Singh Indraprastha University, Delhi, India.
4. Md. Mahfuzul Islam, Alessandro Marchetto, Angelo Susi Fondazione Bruno Kessler Trento, Guiseppe Scanniello, "A Multi-Objective Technique to Prioritize Test Cases Based on Latent Semantic Indexing",16th European Conference on Software Maintenance and Reengineering,2012.
5. Bharti Suri and Shweta Singhal, "Implementing Ant Colony Optimization for Test Case Selection and Prioritization", *International Journal on Computer Science and Engineering (IJCSE)*.
6. Arvind Kumar Upadhyay and A. K. Misra, "Prioritizing Test Suites Using Clustering Approach in Software Testing", *International Journal of Soft Computing and Engineering (IJSCE) ISSN: 2231-2307, Volume-2, Issue-4*, September 2012.
7. Gregg Rothermel, Ronal H. Untch, Chengyun Chu, Mary Jean Harrold, "Test Case Prioritization: An Empirical Study, UK", *Proceedings of the International Conference on Software Maintenance, Oxford*, September 1999.
8. Chien-Li Shen and Eldon Y. Li, "Apply Ant Colony Algorithm to Test Case Prioritization", Department of Information Management, College of Commerce, National Chengchi University, Taiwan.
9. S. Elbaum, A. Malishevsky, and G. Rothermel.(2000), "Prioritizing test cases for regression testing".