A Dynamic Programming Approach for Fault Optimized Sequence Generation in Regression Testing

Monika[1], Dr.Paramjit Singh[2]

[1] Department of Computer Science and Engineering , PDM College of Engineering, Bahadurgarh, Haryana
tuteja.monika710@gmail.com
[2]Professor, PDM College of Engineering, Bahadurgarh, Haryana
director_engg@pdm.ac.in

Abstract- *Delivered software is required to be modified because of some fault, user requirement or because of some new included feature. In such case, when the code of some software is modified, it is also required to test the software again. But instead of testing the complete software again, only few selected test cases are regenerated. It is desired that there should be an effective approach so that an optimized solution for test sequence generation can be found out with low cost. Several researchers have used different techniques for fault optimized and cost effective test sequence generation and one of them is DYNAMIC PRIORITIZATION technique which is used for scheduling test cases in an order so that their effectiveness can be increased at meeting some performance goal. In the proposed work, a two level prioritization approach is recommended for the selection of the test cases and the sequence. In the first layer, the modified code blocks will be analyzed and the relative interaction with the other modules will be analyzed. Based on the number of interacted modules, the first level of prioritization will be done. After that these selected modules will be re analyzed under the criticality parameter. The criticality will be categorized based on the error or the fault type in a specific module. Based on this criticality some cost will be assigned to these test cases. Finally a dynamic programming approach will be implemented to identify the test sequence so that the cost of the regression testing will be minimized. The presented work will give an optimized solution for the test sequence generation with low cost.*

Keywords: *Regression testing, Test case Prioritization, Dynamic Prioritization, Test sequence generation*

1. INTRODUCTION

Regression testing is any type of software testing that seeks to uncover software errors by retesting a modified program. The intent of regression testing is to provide a general assurance that no additional errors were introduced in the process of fixing other problems or modifications of software [Roman,2004]. Regression test suites are often simply test cases that software engineers have previously developed and that have been saved so that they can be used later to perform regression testing [Alexey,2006].Re executing all the test cases requires enormous amount of time thus makes the testing process inefficient. Studies show that regression testing accounts for 80% of the testing costs. The three main approaches to reduce the cost of regression testing include test case selection, test suite minimization and test case prioritization.

1.1 Key Issues in Regression Testing

Many issues can be addressed in the context of Regression testing to reduce the cost of regression testing. A lot of challenges and problems are posing a great threat to the technology. Here some most important issues are presented that have dominated the field of research since the evolution of Regression testing.

A. *Regression Test Selection*

Regression test selection techniques select a subset of valid test cases from an initial test suite(T) to test that the affected but unmodified parts of a program continue to work correctly. Use of an effected regression test selection technique can help to reduce the testing costs in environments in which a program undergoes frequent modifications. Regression test selection essentially consists of two major activities:

- Identification of the affected parts –This involves identification of the unmodified parts of the program that are affected by the modifications.
- Test case selection- This involves identification of a subset of test cases from the initial test suite T which can effectively test the unmodified parts of the program [Swarnendu , 2011].

B. *Test Case Minimization*

Test case minimization means reducing the test suite size to a minimal subset to maintain the same level of coverage as the original test suite. There is empirical evidence indicating that fault detection capabilities of test suites can be severely compromised by minimization [Rothermel, 1998]. The significance of minimization is that the resulting minimized set has the same coverage with respect to a certain criterion (say C) as the original set. We do minimization only on a subset of regression tests determined by using the modification based test selection technique. The advantages are (1) reducing the amount of work required by test set minimization (2) having a higher chance to select more tests on which the new and the old programs produce different outputs, and (3) having a lesser chance to include test cases that fail to distinguish the new program from the old [Eric,1997].

C. *Test Case Prioritization*

Test case prioritization techniques prioritize and schedule test cases in an order that attempts to maximize some objective function .For example, software test engineers might wish to schedule test cases in an order that achieves code coverage at the fastest rate possible, exercises features in order of expected frequency of use, or exercises sub systems in an order that reflects their historical propensity to fail. When the time required to run all the test cases in test suite is sufficiently long, the benefits offered by test case prioritization methods become more significant [Siripong,2010].

D. *Sequence Generation*

To conduct efficient and effective regression testing, a test case sequence is generated from the existing regression suite. First, construct a superset of all regression tests that should be used to ensure that a new program preserves the desired functionality of the old program .Such construction is done by a modification based test selection. Second, if necessary, use prioritization or minimization for further test screening based on those selected by modification [Eric,1997].

E. *Obsolete, Retestable and Redundant Test Cases*

Test cases in the initial test suite can be classified as obsolete , retest able and redundant test cases. Obsolete test cases are no more valid for the modified program. Retest able test cases are those test cases that execute the modified and the affected parts of the program and need to be rerun during the regression testing. Redundant test cases execute only the unaffected parts of the program. Although these are valid test cases ,these can be omitted from the regression test suite without compromising the quality of testing [Leung,1989].

F. *Fault-revealing Test Cases*

A test case $t \in T$ is said to be *fault-revealing* for a program P, if it can potentially cause P to fail by producing incorrect outputs for P [Rothermel,1996].

G. *Modification-traversing Test Cases*

A test case $t \in T$ is *modification-traversing* for P and P', if the execution traces of t on P and P' are different [Rothermel,1996]. In other words, a test case t is said to be modification-traversing if it executes the modified regions of code in P'. For a given original program and its modified version, the set of modification-traversing test cases is a super-set of the set of the modification-revealing test cases.

1.2 Need of Sequence Generation in Regression Testing

In above we discussed the basics of Regression testing like introduction and key issues but the main important point here is sequence generation and is one of the main challenges of regression testing. It is noticed that several companies have "constant test cases set" for regression testing and they are executed irrespective of the number and type of bug fixes. Sometimes this approach may not find all side effects in the system and in some cases it may be observed that the effort spent on executing test cases for regression testing can be minimized if some analysis is done to find out what test cases are relevant and what are not. To conduct efficient and effective regression testing, a test case sequence is generated from the existing regression suite. First, construct a superset of all regression tests that should be used to ensure that a new program preserves the desired functionality of the old program . Second, if necessary, use prioritization or minimization for further test screening based on those selected by modification [Eric,1997].

2. REGRESSION TESTING APPROACHES

A number of different approaches have been studied to aid the regression testing process. The three major branches include test suite minimization, test case selection and test case prioritization. Test suite minimization is a process that seeks to identify and then eliminate the obsolete or redundant test cases from the test suite. Test case selection deals with the problem of selecting a subset of test cases that will be used to test the changed parts of the software. Finally, test case prioritization concerns the identification of the 'ideal' ordering of test cases that maximizes desirable properties, such as early fault detection [Yoo,2007].

2.1 *Test Suite Minimization Approach*

Test case minimization techniques reduce the test suite size to a minimal subset to maintain the same level of coverage as the original test suite. There is empirical evidence indicating that fault detection capabilities of test suites can be severely compromised by minimization [Rothermel,1998]. The significance of minimization is that the resulting minimized set has the same coverage with respect to a certain criterion (say C) as the original set. We do minimization only on a subset of regression tests determined by using the modification based test selection technique. The advantages are (1) reducing the amount of work required by test set minimization (2) having a higher chance to select more tests on which the new and the old programs produce different outputs, and (3) having a lesser chance to include test cases that fail to distinguish the new program from the old [Eric,1997]. Test suite minimization techniques aim to identify redundant test cases and to

remove them from the test suite in order to reduce the size of the test suite.

2.2 *Regression Test Selection Approach*

Regression test selection techniques select a subset of valid test cases from an initial test suite(T) to test that the affected but unmodified parts of a program continue to work correctly. Use of an effected regression test selection technique can help to reduce the testing costs in environments in which a program undergoes frequent modifications. Regression test selection essentially consists of two major activities:

- Identification of the affected parts –This involves identification of the unmodified parts of the program that are affected by the modifications.
- Test case selection- This involves identification of a subset of test cases from the initial test suite T which can effectively test the unmodified parts of the program. The aim is to be able to select the subset of test cases from the initial test suite that has the potential to detect errors induced on account of the changes [Swarnendu,2011].The use of an RTS technique can reduce the cost of regression testing compared to the retest- all approach. The retest –all approach is considered impractical on account of cost resource and delivery schedule constraints that projects are frequently subjected to [Leung,1991].

2.3 *Test Case Prioritization Approach*

Test case prioritization techniques prioritize and schedule test cases in an order that attempts to maximize some objective function. For example, software test engineers might wish to schedule test cases in an order that achieves code coverage at the fastest rate possible, exercises features in order of expected frequency of use, or exercises subsystems in an order that reflects their historical propensity to fail. When the time required to execute all test cases in a test suite is short, test case prioritization may not be cost effective - it may be most expedient simply to schedule test cases in any order. When the time required to run all test cases in the test suite is sufficiently long, the benefits offered by test case prioritization methods become more significant [Dennis, 2006]. Although test case prioritization methods have great benefits for software test engineers, there are still outstanding major research issues that should be addressed. The examples of major research issues are: (a) existing test case prioritization methods ignore the practical weight factors in their ranking algorithm (b) existing techniques have an inefficient weight algorithm and (c) those techniques are lack of the automation during the prioritization process. Test case prioritization techniques provide a way to schedule and run test cases, which have them highest priority in order to provide earlier detect faults [Siripong, 2010]. Some techniques related to prioritization approach are as follows:

2.3.1 *Customer Requirement-Based Prioritization Techniques*

Customer requirement-based techniques are methods to prioritize test cases based on requirement documents. many weight factors have been used in these techniques, including custom-priority, requirement complexity and requirement volatility. Test case prioritization techniques and use of several factors to weight (or rank) the test cases are introduced. Those factors are the customer-assigned priority (CP), requirements complexity(RC) and requirements volatility (RV). Additionally, values are assigned (1 to 10) for each factor for the measurement. Here higher factor values indicate a need for prioritization of test case related to that requirement [Jeffery,1999].

Weight prioritization (*WP*) measures the important of testing a requirement earlier.

$WP = \Sigma\ (PF_{value} * PF_{weight})$; PF=1 to n (1)

Where:

- *WP* denotes weight prioritization that measures the importance of testing a requirement.
- PF_{value} is the value of each factor, like CP, RC and RV.
- PF_{weight} is the weight of each factor, like CP, RC and RV.

Test cases are then ordered such that the test cases for requirements with high WP are executed before others [Siripong,2010].

Two particular goals of test case prioritization approaches: (a) to improve user perceived software quality in a cost effective way by considering potential defect severity and (b)to improve the rate of detection of severe faults during system level testing of new code and regression testing of existing code [Lehmann,2000].

General test case prioritization technique and associated metric based on varying testing requirement priorities and test case costs. An algorithm is proposed that weights test cases by the following factors: (a) test history (b) additional requirement coverage (c) test case cost and (d) total requirement coverage [Xiaofang,1988].

.

2.3.2 *History-based Approach*

A prioritization technique based on association clusters of software artifacts obtained by a matrix analysis called singular value decomposition [Sherriff, 2007]. The prioritization approach depends on three elements: association clusters, relationship between test cases and files and a modification vector. Association clusters are generated from a change matrix using SVD; if two files are often modified together as a part of a bug fix, they will be clustered into the same association cluster. Each file is also associated with test cases that affect or execute it. Finally, a new system modification is represented as a vector in which the value indicates whether a specific file has been modified. Using the association clusters and the

modification vector, it is then possible to assign each file with a priority that corresponds to how closely the new modification matches each test case. One novel aspect of this approach is that any software artifact can be considered for prioritization. Sherriff et al. noted that the faults that are found in non-source files, such as media files or documentation, can be as severe as those found in source code [Yoo,2007].

2.3.3 *Coverage-based Techniques*

Test coverage analysis is a measure used in software testing known as code coverage analysis for practitioners. It describes the quantity of source code of a program that has been exercised during testing. It is a form of testing that inspects the code directly and is therefore a form of white box testing. The following lists a process of coverage-based techniques: (a) finding areas of a program not exercised by a set of test cases (b) creating additional test cases to increase coverage (c) determining a quantitative measure of code coverage, which is an indirect measure of quality and (d) identifying redundant test cases that do not increase coverage. The coverage-based technique is a structural or white-box testing technique. Structural testing compares test program behavior against the apparent intention of the source code. This contrasts with functional or black-box testing, which compares test program behavior against a requirements specification. It examines how the program works, taking into account possible pitfalls in the structure and logic. Functional testing examines what the program accomplishes, without regard to how it works internally. The coverage based techniques are methods to prioritize test cases based on coverage criteria, such as requirement coverage, total requirement coverage, additional requirement coverage and statement coverage [Siripong,2010].

2.3.4 *Cost Effective-Based Prioritization Techniques*

Cost effective-based techniques are methods of prioritizing test cases based on costs, such as cost of analysis and cost of prioritization. Many researchers have researched this area. The following paragraphs present existing cost effective-based test case prioritization techniques. The cost of a test case is related to the resources required to execute and validate it. Additionally, cost-cognizant prioritization requires an estimate of the severity of each fault that can be revealed by a test case. Four practical code coverage- based heuristic techniques are: total function coverage prioritization (*fn-total*), additional function coverage prioritization (*fn-addtl*), total function difference-based prioritization (*fn-diff-total*) and additional function difference-based prioritization(*fn-diff-addtl*) [Alexey,2002].
Cost models for prioritization that take these costs into account. They defined the following variables to prioritize test cases: cost of analysis, Ca(T) and cost of the prioritization algorithm, Cp(T).

$$WP = Ca(T) + Cp(T)$$

Where:

- *WP* is a weight prioritization value for each test case.
- *Ca(T)* includes the cost of source code analysis, analysis of changes between old and new versions, and collection of execution traces.
- *Cp(T)* is the actual cost of running a prioritization tool, and, depending on the prioritization algorithm used, can be performed during either the preliminary or critical phase [Alexy,2002].

3. PROPOSED WORK

The proposed work is about the selection of the test sequence based on some defined constraints. At the initial level, we need to define a database to maintain all the code modules along with test cases respective to the project. These code modules will be defined respective to the changes occur in the module as well the relation with other modules. This module interaction analysis is the important constraint for the selection of the test case for regression testing.
Once the test cases are identified, the next work is about to analyze the module criticality. The criticality will be analyzed based on the number of faults and the type of faults. Along with these two parameters the cost of the relative test case generation will be defined.
The prioritization would be done according to the criticality of the test as well as the code on which the test is occurred. Finally, the test sequence will be generated by using the dynamic programming approach. The test sequence will be identified with minimum test cost.

3.1 *Overall Design*

The overall design of the proposed work is given as under:

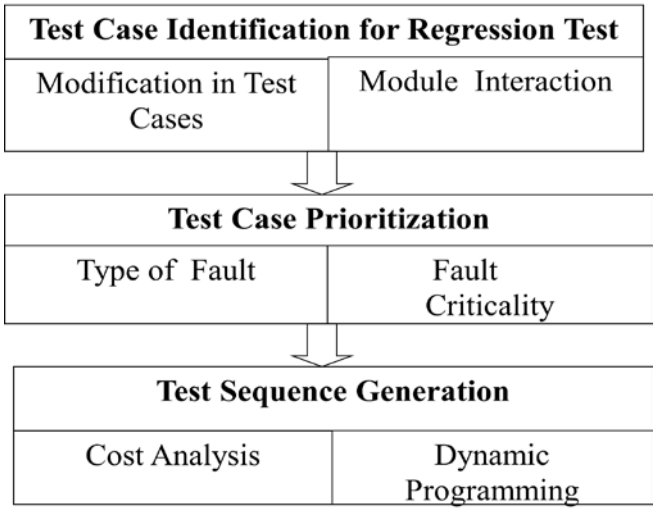
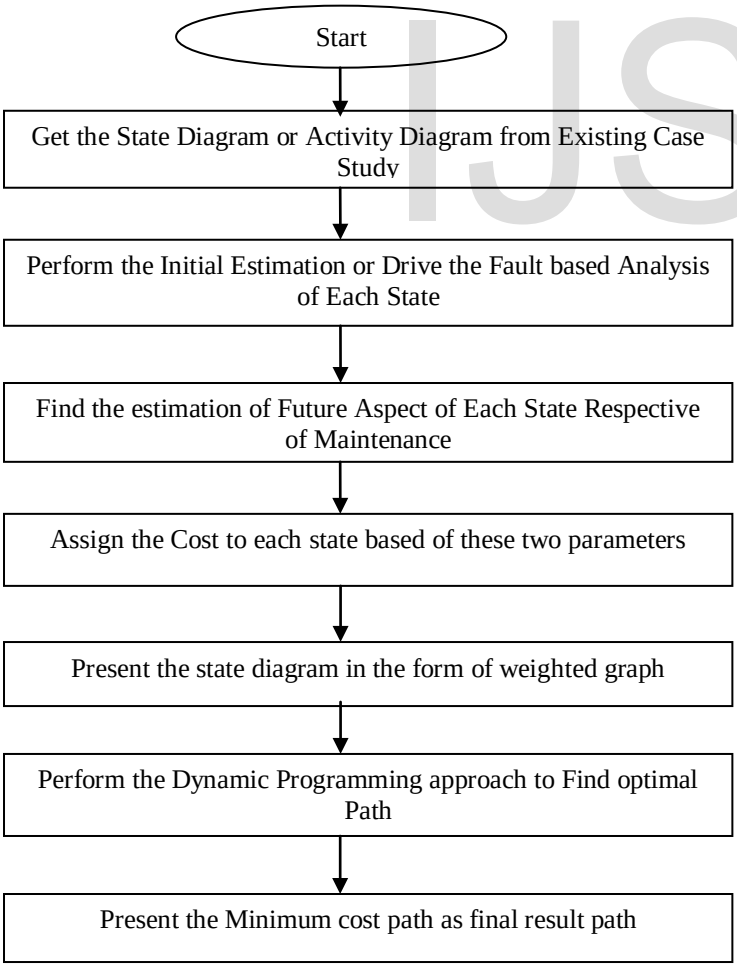


Figure 1 Overall Design

3.1.1 *Work Flow*

The below diagram shows the work flow for the proposed work:



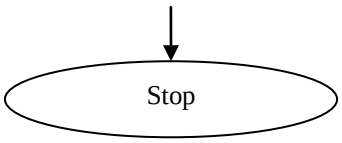


Figure 2 Work Flow

4. RESULTS

4.*1 Example Program*

```
Void main()
{
Int a,b,c;
Printf("Enter number");
Scanf("%d%d",&a,&b);
C=sqrt(a)/b;
Printf("%d",c);
}
```

4.1.1 *Generated Test Cases*

Test Cases

1. Variable A Assigned Some Value;
2. Variable B Assigned Some Value;
3. Variable C Assigned Some Value;
4. Check A for 0
5. Check B for 0
6. Check A for –ve value
7. Check B for –ve value
8. Check for input specifier for A,B
9. Check for Expression Result of C
10. check for output Specifier for C

4.1.*2 Prioritization*

Critical Test cases
Here in given program Test Case 5 will return failure as of B is 0
Here Test Case 6 will return Error if A is I –ve
Here Test Case 9 will return error if any of Case 5 or 6 is false
Case 10 will not return correct result if specifier is wrong

4.1.3 *Approach*

Preliminary Check

1. No Variable is unassigned (CASE 1,2,3)
2. Check format specifier for all values (Case 8,10)

Static Test

1 No Variable is unassigned (CASE 1,2,3)
2 Check format specifier for all values (Case 8,10)

Dynamic Test

All other Test Cases are dynamic

4.2 *Results*

As the general case we have assigned the random cost to each test case and perform the analysis based on this random cost assignment. The output driven based on this assignment is shown as under.

a) The obtained Test Sequence of this random cost assignment is given as

3 9 4 1 10 7 8 5 6 2

b) The cost driven from the on given approach is given as

Cost = 11.0547

Sequence can be

1, 2, 3,8,10
4, 5,6,7,9

Modified Sequence will be

- Assignment Test
- Format specifier test
- Input values test
- Output values test

Here are some graphs showing the results.

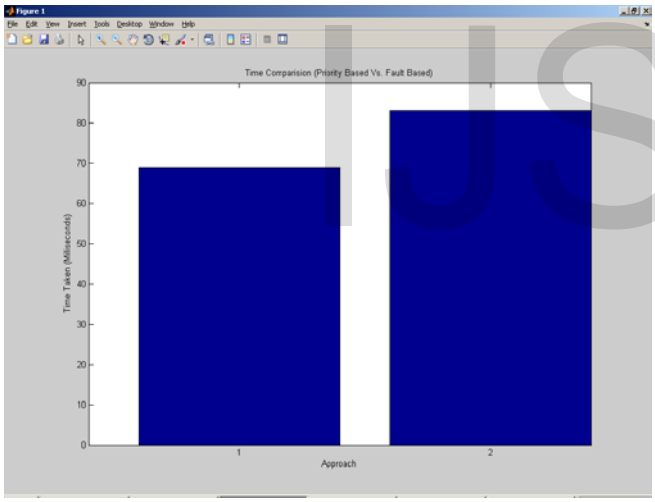


Figure 3: Cost Analysis (Proposed Vs. Existing)

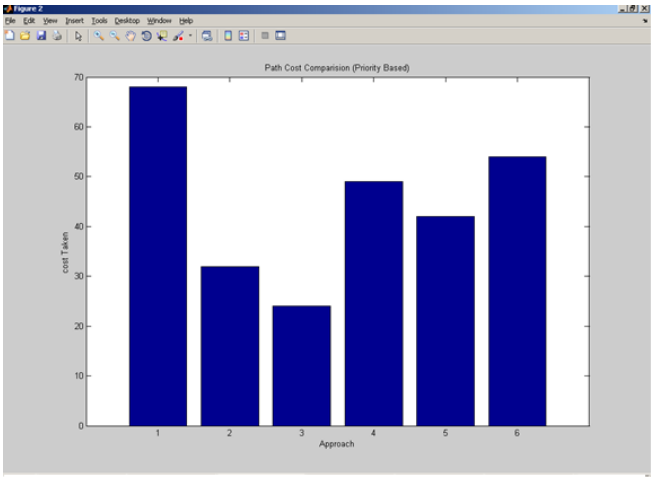


Figure 4 : Cost Analysis (Proposed Approach)

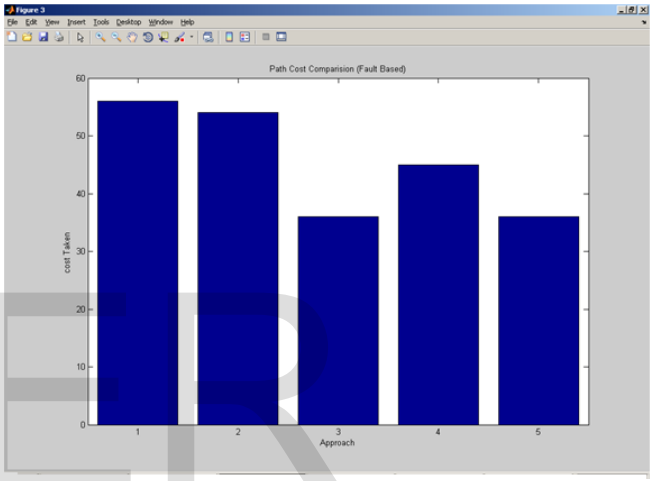


Figure 5: Cost Analysis (Proposed Approach)

Here figure is showing the comparative analysis of different cost testing path cost in case of fault based test path estimation.

5. CONCLUSION AND FUTURE WORK

In this present work we have improved the existing path testing approach by implementing the dynamic programming approach. A software project is the sequence of correlated code modules where some test cases are associated with each module. In this work, we have defined these test cases respective to the fault occurrence parameter. According the importance of test cases, some priority value is assigned to each test case. After this the dynamic programming approach is implemented to find the best path respective to the low cost and less chances of fault occurrence during the testing process.

The main parameter taken for the research work is chances of fault occurrence. The work can be extended in different directions:

1. We can also use some more optimization approaches to get the better results such as Genetics or the Neural Network
2. We can also use some different parameters to estimate the test path cost such as number of connections with next and previous code modules etc

References

[1] **Alexey G. Malishevsky, Joseph R. Ruthruff, Gregg Rothermel, Sebastian Elbaum**, Cost-cognizant Test Case Prioritization," Technical Report TR-UNL-CSE-2006-004, *Department of Computer Science and Engineering, University of Nebraska–Lincoln, Lincoln, Nebraska, U.S.A.*, March 2006.

[2] **Alexey G. Malishevsky, Gregg Rothermel and Sebastian Elbaum**, Modeling the Cost-Benefits Tradeoffs for Regression Testing Techniques, *Proceedings of the International Conference on Software Maintenance (ICSM'02)*, 2002.

[3] **Dennis Jeffrey and Neelam Gupta**, Test Case Prioritization Using Relevant Slices , In *Proceedings of the 30th Annual International Computer Software and Applications Conference*, Volume 01, 2006, pages 411-420, 2006.

[4] **Eric W. Wong, J. R. Horgan, Saul London, Hira Agrawal** *Bell Communications Research 445 South Street Morristown*, NJ 07960.

[5] **Jeffery von Ronne**, Test Suite Minimization: An Empirical Investigation, 1999.

[6] **Korel B. and Laski J**., Algorithmic software fault localization , *Annual Hawaii International Conference on System Sciences*, pages 246–252, 1991.

[7] **Lehmann E. and Wegener J**., Test case design by means of the CTE XL, *In Proc. of the 8th European International Conf. on Software Testing, Analysis & Review* (EuroSTAR 2000), 2000.

[8] **Leung H. and White L**., A cost model to compare regression test strategies. In *Proceedings of the Conference on Software Maintenance*, pages 201–208, 1991.

[9] **Onoma K., W.-T. Tsai, M. Poonawala, and H.Suganuma**, Regression testing in an industrial environment, *Comm. Of the ACM*, 41(5):81–86,1988.

[10] **Rothermel G., M. J. Harrold, J. Ostrin, and C. Hong**, An empirical study of the effects of minimization on the fault detection capabilities of test suites ,in *Proc. of the International Conference on Software Maintenance*, 1998, pp. 34–43.

[11] **Savenkov R**.,How to become a software tester. (*Roman Savenkov Consulting*, 2004)

[12] **Sherriff M, Lake M, Williams L**. Prioritization of regression tests using singular value decomposition with empirical change records. *Proceedings of the The 18th IEEE International Symposium on Software Reliability* (ISSRE 2007), IEEE Computer Society: Washington, DC, USA, 2007; 81–90.

[13] **Siripong Roongruangsuwan, Jirapun Daengdej**," Test case prioritization techniques", *Autonomous System Research Laboratory, Science and Technology, Assumption University, Thailand.*(2010).

[14] **Swarnendu Biswas and Rajib Mall** ,*Dept. of Computer Science and Engineering IIT Kharagpur*, India - 721302

[15] **Yoo S., M. Harman**, "Regression testing Minimisation, selection and prioritization, King's College London, *Centre for Research on Evolution, Search & Testing, Strand, London, WC2R 2LS, UK*(2007)