Regression Based Prioritization Testing Using Graphical User Interface

Jyoti[1], Kirti Bhatia[2]
[1]*Mtech Scholar, Sat Kabir Institute of technology & Management, Bahadurgarh*
[2]*Assistant Professor, Sat Kabir Institute of technology & Management, Bahadurgarh*

Abstract—Regression is done in two situations 1) If software has been changed (because of fixes or Adding extra functionality or deleting existing functionality, 2) If the Environment changes still we will do regression. Regression testing will be conducted after any bug fixed or any functionality changed.Test suite minimization technique address this issue by removing redundant test cases and Test case prioritization technique by scheduling test cases in an order that enhance the efficiency of attaining some performance criteria. This paper presents a new approach for regression testing by combining these two techniques. The approach is to first minimize the test suite by using greedy approach and then prioritize this minimized test suite using genetic algorithm. Proposed approach supports tester by minimizing the test suite while ensuring all the requirement coverage and minimum execution time. The overall aim of this research is to make the testing process time and cost effective by reducing the number of test cases that need to run after changes have been made.

Keywords—prioritization technique, regression testing, efficiency

I. INTRODUCTION

Regression testing is a type of software testing that verifies that software previously developed and tested still performs correctly after it was changed or interfaced with other software.Changes may include software enhancements, patches, configuration changes, etc. During regression testing, new software bugs or regressions may be uncovered. Sometimes a software change impact analysis is performed to determine what areas could be affected by the proposed changes. These areas may include functional and non-functional areas of the system.Retesting (also called Confirmation testing) when testing is done to confirm that the bug which we reported earlier has been fixed in new build is called retesting but Regression testing means testing to ensure that the fixes has not introduced new bugs to beappear.

When a bug is fixed by the development team than testing the other features of the applications which might be affected due to the bug fix is known as regression testing. Regression testing is always done to verify that modified code does not break the existing functionality of the application and works within the requirements of the system.

For Example there are three Modules in the Project named Admin Module, Personal Information, and Employment Module and suppose bug occurs in the Admin Module like on Admin Module existing User is not able to login with valid login credentials so this is the bug. Now Testing team sends the above - mentioned Bug to the Development team to fix it and when development team fixes the Bug and hand over to Testing team than testing team checks that fixed bug does not affect the remaining functionality of the other modules (Admin, PI, Employment) and also the functionality of the same module (Admin) so this is known as the process of regression testing done by Software Testers.

1. Retest all – It is one of the conventional techniques for regression testing in which each and every test case in the existing test suite are retuned. This technique is not feasible most of time as it require more time and budget.
2. Regression Test Selection (RTS) - Here, RTS allows us to omit some of the test cases. RTS is beneficial only if the cost of selecting some of the test cases is less than the cost of executing the complete test suite. In this technique only part of test cases in test suite is selected to rerun. RTS techniques are further classified into three categories [3]CoverageTechnique, Minimization Technique and Safe Technique.
3. Test Case Prioritization – Test case prioritization techniques arranges test cases in a most beneficial order thus making the testing process more effective. There are 18 different test case prioritizations techniques [4] numbered P1-P18 which is divided into three groups:-Comparator Techniques, Statement Level Techniques, and Function Level Techniques.

II. REGRATION TESTING AND STRATEGIES

Regression testing is an expensive and lavish process in which a number of approaches of regression testing are utilized to enhance its viability and improve effectiveness as it may accounts 70% of the total cost. Techniques used to improve its effectiveness are retest all, test case selection, test case reduction and test case prioritization. Retest all approach involves retesting and re-execution of all test cases defined in test suites, which are more cost effective due to their critical behavior and severity. In order to increase cost effectiveness and efficiency, test case prioritization approach is used to prioritize the test cases by rearranging.

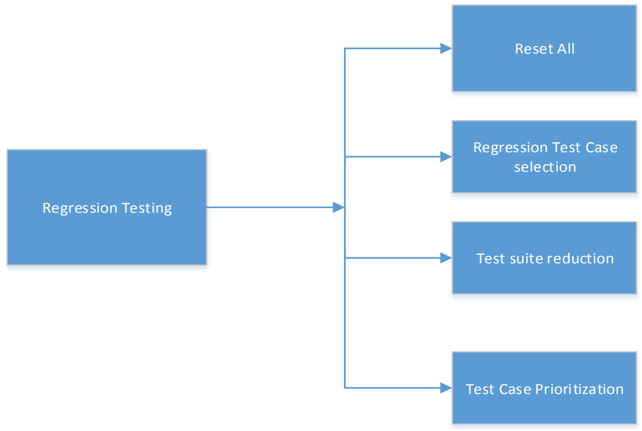


Fig.1. Regression Testing Techniques

Regression testing is achieved after the bug fixed, means testing the operation whether the fixed defect is affecting remaining functionality of the application or not. Usually in regression testing bug fixed module is tested. During regression testing tester always check the entire system whether the fixed bug make any adverse effect in the existing system or not.

There are mostly three strategies to regression testing, 1) to run all tests called unit Regression test 2) partial regression Test and 3) always run a subset of tests based on a test case prioritization technique called full regression test.

By increasing the overall rate of severe fault detection, a greater number of errors can be found more rapidly in the code developed to meet user requirements. As frequent rebuilding and regression testing achieves popularity, the need for a time constraint aware prioritization technique developing as per requirements. New software development processes such as extreme programming also promote a short development and testing cycle and frequent execution of fast test cases. Therefore, there is a clear need for a prioritization technique that has the potential for more effectiveness when a test suite's allowed execution time is known, particularly when that execution time is short.

III. Mythology for Regression Testing

The purpose of regression testing is to ensure that changes such as those mentioned above have not introduced new faults. One of the main reasons for regression testing is to determine whether a change in one part of the software affects other parts of the software. Common methods of regression testing include rerunning previously completed tests and checking whether program behavior has changed and whether previously fixed faults have re-emerged. Regression testing can be performed to test a system efficiently by systematically selecting the appropriate minimum set of tests needed to adequately cover a particular change.Regression testing is the re-testing of features to make safe that features working earlier are still working fine as desired.

It is executed when any new build comes to QA (quality accurance), which has bug fixes in it or during releasing cycles (Alpha, Beta or GA(Genetic algorithm) to originate always the endurance of product.

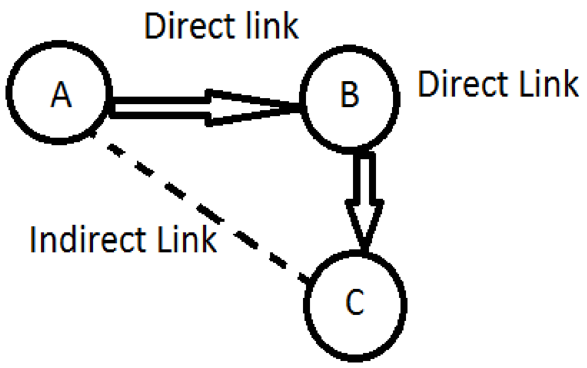


Fig.2. Regression Testing Techniques Approach

Let A, B, C are the three modules where A can be direct interaction with B and B can direct interact with C.in this case C module interconnect with A indirectly. Based on the testing can divide in three form. 1) Unit regression testing, 2) partial testing and 3) full regression or prioritization technique for testing.

A. *Unitregression testing*

Unit Regression is done only when a defect is fixed in a module, if that module has relationship with other sections within that Module, then we perform Unit Regression. We will be testing only the fixed part of the module, if any defect is fixed. This happens when the Developer who has fixed the defect says to perform the Unit Regression.The goal of unit testing is to isolate each part of the program and show that the individual parts are correct. A unit test provides a strict, written contract that the piece of code must satisfy. As a result, it affords several benefits.

B. *Partial regression testing*

Partial regression is when regression will be done after Impact Analysis and we would go for Partial regression means complete Regression suite will not be executed.

C. *Prioritization testing*

Testing the changes and all the remaining features, Normally done when the changes are done in the root of the product or whenever the modifications or changes are more in the product we do full regression testing.

Full regression or Test case prioritization techniques organize the test cases in a test suite, prioritize them increase in the effectiveness of testing on the basis of requirements. One performance goal, the severe fault detection rate, is a measure of how quickly severe faults are detected during the testing process. An improved rate of severe fault detection can provide faster feedback regarding the quality of the system under testing, as complete testing process is vast and too expensive. This is often the case with regression testing, the process of validating the modified version of software to detect whether new errors have been introduced into previously tested code and to provide confidence that modifications are correct.

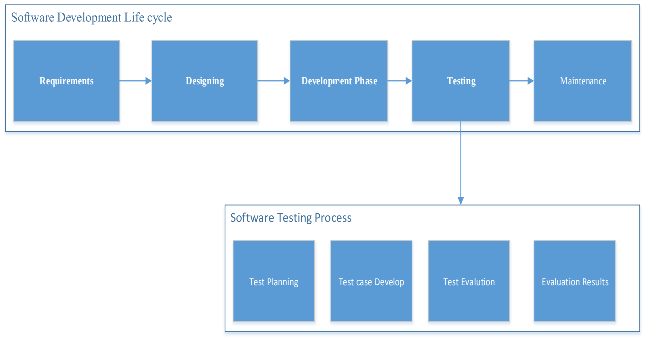


Fig.3: Software Development Life Cycle

IV. APPROACH AND ALGORITHM

The time constrained test case prioritization problem can be effective and reduced to the NP-complete zero/one knapsack problem. This can often be efficiently approximated with a genetic algorithm (GA) heuristic search technique. Genetic algorithms have been effectively used in other software engineering and programming language problems such as test generation, program transformation, and software maintenance resource allocation, this survey demonstrates that they also prove to be effective in creating time constrained test prioritizations using requirements factors and technique that prioritizes regression test suites so that the new ordering:-

1. Will always run within a given time limit.
2. Will have the highest possible potential for severe defect detection based on derived coverage information and requirements.

In summary, the important contributions of this survey are as follows:

a) A GA based technique to prioritize a regression test suite that will be run within a time constrained execution environment.

b) An empirical evaluation of the effectiveness of the resulting prioritizations in relation to (i) GA-produced prioritizations using different user requirement parameters.

From the above, the testing phases contain the following processes: test planning, test development, test execution and evaluation of results.With existing test case prioritization techniques researched in 1998-2015, this paper introduces and organizes a new “4C” classification of those existing techniques, based on their prioritization algorithm’s characteristics, as follows:

1. Customer Requirement-based techniques:-Customer requirement-based techniques are methods to prioritize test cases based on requirement documents. Also, many weight factors have been used in these techniques, including custom-priority, requirement complexity and requirement volatility.
2. Coverage-based techniques: 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.
3. Cost Effective-based techniques: Cost effective-based techniques are methods to prioritize test cases based on costs, such as cost of analysis and cost of prioritization.
4. Chronographic history-based techniques: Chronographic history-based techniques are methods to prioritize test cases based on test execution history.

Test cases in the test suites are reschedule which will further prioritize using an algorithm. Before dealing with prioritization algorithms the problem associated with test case prioritization requires understanding which is defined as follows:

Dentition 1: Given: T, a test suite, PT, the set of permutations of T , and f , a function from PT to the real numbers.

Problem: Find T’ ϵ PT such that (ΛT’’) (T’’ ϵ PT) (T’’ $\neq$ T’) [f (T’) $\geq$ f (T’’)]

Prioritization Factor Value (pfv) = PFvalue * PFweight

In this definition, PT represents the set of all possible prioritizations (orderings) of T , and f is a function that, applied to any such ordering, yields an award value for that ordering. (F or simplicity, and without loss of generality, the definition assumes that higher award values are preferable to lower ones.)

V. PROPOSED TEST CASE PRIORITIZATION APPROACH USING GENETIC ALGORITHM

Genetic algorithm is stochastic search technique, which is based on the idea of selection of the fittest chromosome. In genetic algorithm, population of chromosome is represented by different codes such as binary, real number, permutation etc. genetic operators(i.e. selection, crossover, mutation) is applied on the chromosome in order to find more fittest chromosome.

The fitness of a chromosome is defined by a suitable objective function. As a class of stochastic method genetic algorithm is different from a random search. While genetic algorithm carry out a multidimensional search by maintaining population of potential user, random methods consisting of a combination of iterative search methods and simple random search methods can find a solution for a given problem. One of the genetic method's most attractive features is to explore the search space by considering the entire population of the chromosome.

The Genetic algorithm is an evolutionary algorithm and population based search method. The selection takes place from the available population using fitness function; genetic operators are applied to obtain an optimal solution, followed by termination.

The steps of genetic algorithm are as-

1. Generate population (chromosome).
2. Evaluate the fitness of generated population.
3. Apply selection for individual.
4. Apply crossover and mutation.
5. Evaluate and reproduce the chromosome.

Generate population (chromosome): Initially population is randomly selected and encoded. Each chromosome represent the possible solution of the problem.(in our case the sequence of test cases is chromosome and our aim is to optimize this sequence).
For example- for 12 test cases T1, T2, T3……….T12 the sequence is
T1->T2->T4->T6->T9->T10->T12->T3->T5->T7->T8->T11

Evaluate the fitness of generated population: The fitness of a chromosome is defined by an objective function. An objective function tells how 'good' or 'bad' a chromosome is. This objective function generates a real number from the input chromosome. Based on this number two or more chromosome can be compared.

Apply selection for individual: In general the selection is depending on the fitness value of the chromosome. The chromosome with higher or lower value will be selected based on the problem definition.

Apply crossover and mutation: Parents are choosing and randomly combined. This technique for generating random chromosome is called crossover. There exist two type of crossover a Single point crossover.

A. *Multiple point crossovers*

For example- suppose two sequences for test cases is

TABLE I
TYPE OF FAULT

Code	Mnemonic	Description
M1	unordered	no prioritization (control)
M2	random	randomized ordering
M3	optimal	ordered to optimize rate of fault detection
M4	stmt-total	prioritize in order of coverage of statements
M5	stmt-addt	prioritize in order of coverage of statements not y et covered
M6	branch-total	prioritize in order of coverage of branches
M7	branch-addtl	prioritize in order of coverage of branches not y et covered
M8	FEP-total	prioritize in order of total probability of exposing faults
M9	FEP-addtl	prioritize in order of total probability of exposing faults, adjusted to consider effects of previous test cases

P1: T1->T2->T3->T4->T5->T6->T7->T8->T9
And
P2: T4->T2->T5->T7->T8->T1->T6->T9->T2
Then using one point crossover offspring will be-
C1: T1->T2->T3->T4->T8->T6->T9->T5->T7
C2: T4->T3->T5->T7->T6->T8->T9->T1->T2

For C1 write first part of the P1 as it is and thenwrite second part of P2 with constraint that a test case has not been added in to C1.For doing mutation two genes selected randomly along the chromosome and swapped with each other.
~~For example- when T3 and T9 get selected randomly~~
T1->T2->T3->T4->T8->T6->T9->T5->T7
T1->T2->T9->T4->T8->T6->T3->T5->T7

Termination criteria: The termination criteria can be selected in the different ways such as- reaching the predefined fitness value, the number of generation or a nonexisting difference in the fitness values of each generation.In our approach we used a fixed generation number as a termination criteria.
The goal of increasing the likelihood of revealing faults earlier in the testing process. Informally, we describe this goal as one of improving our test suite's r ate of fault detection: we describe a function f that quantizes this goal.

There are several aspects of the test case prioritization problem that are worth describing further. First, there are many possible goals of prioritization, including the following:

There are several motivations for meeting this goal. An improved rate of fault detection during regression testing can let software engineers begin their debugging activities earlier than software. An improved rate of fault detection can also provide faster feedback on the system under test, and provide earlier evidence when quality goals have not been met, allowing strategic decisions about release schedules to be made earlier than might otherwise be possible. Further, in a testing situation in which the amount of testing time that will be available is uncertain (for example, when mark et pressures ma y force a release of the pro duct prior to execution of all test cases), such prioritization can increase the likelihood that whenever the testing process is terminated, testing resources will have been spent more cost-effectively in relation to potential fault detection than they might otherwise have been. In this paper, we consider nine different test case prioritization techniques. Table 1 lists these techniques; we next present the techniques in turn.

VI. SIMULATION RESULTS

In this section we present technique to make regression testing efficient. Let's say a program, P has a test suite T in the form of Test case- requirement matrix, representing the requirements each test case is covering as shown in Table 1.

TABLE 2

Test suite for program p

Test Case	Requirement						
	S1	S2	S3	S4	S5	S6	S7
T1	1	0	1	1	0	0	0
T2	0	1	1	1	1	0	0
T3	1	1	1	1	0	0	0
T4	0	0	0	0	1	0	1
T5	1	0	0	1	0	1	0
T6	1	0	1	0	0	0	0
T7	0	1	0	1	0	1	0
T8	0	0	0	0	0	0	0
T9	0	1	0	1	1	1	0

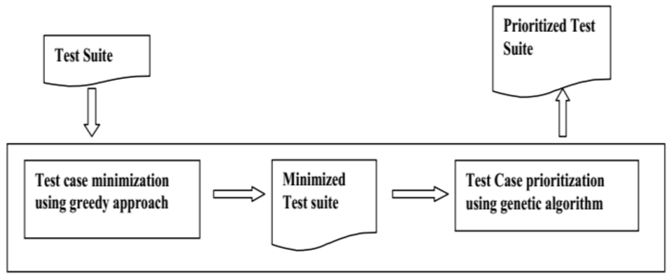
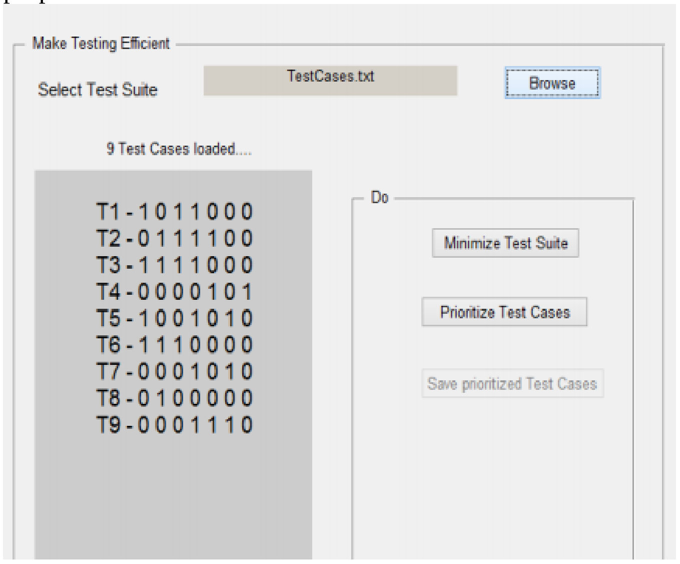

Test Suite
Prioritized Test Suite
Test case minimization using greedy approach
Minimized Test suite
Test Case prioritization using genetic algorithm

Fig.4. Overall view of our approach

The experiment is implemented in MATLAB. The test case prioritization technique's basic evaluation is to have maximum number of faults covered and statement covered with minimum number of test cases required. In this approach, the execution time of every test case is also analyzed. The fault measuring technique used in fault coverage is based testing technique. In this example, there are test cases forming Test Suite (TS) = {T1, T2, T3, T4, T5, T6, T7, T8, T9, } Similarly the statements covered by the test cases are denoted as Statements Covered (SC) = {S1, S2, S3, S4, S5, S6, S7}. The Control Flow Graph (CFG) is seen in figure 1

We are defining a new approach to minimize and then priorities the test cases using genetic algorithm. For this purpose we have used the MATLAB simulator.

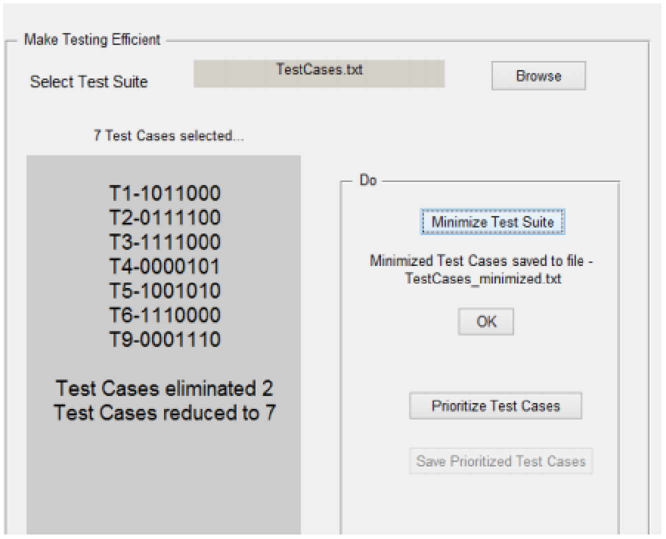


9 Test Cases loaded....

```
T1-1011000
T2-0111100
T3-1111000
T4-0000101
T5-1001010
T6-1110000
T7-0001010
T8-0100000
T9-0001110
```

Fig.5. Prepared Test Suite

This greedy prioritization algorithm may not always choose the optimal test case ordering. To see this, suppose a program contains four faults, and suppose our test suite for that program contains three test cases that detect those faults as shown in Table 2



7 Test Cases selected...

```
T1-1011000
T2-0111100
T3-1111000
T4-0000101
T5-1001010
T6-1110000
T9-0001110

Test Cases eliminated 2
Test Cases reduced to 7
```

Minimized Test Cases saved to file - TestCases_minimized.txt

Fig.6. Minimized Test Suite

For prioritizing these minimized Test Suite a random cost is assigned to each test case . Prioritized output for this random assignment is shown in fig 5. The basis paramenters defined while performing Genetic Algorithm for prioritization are:

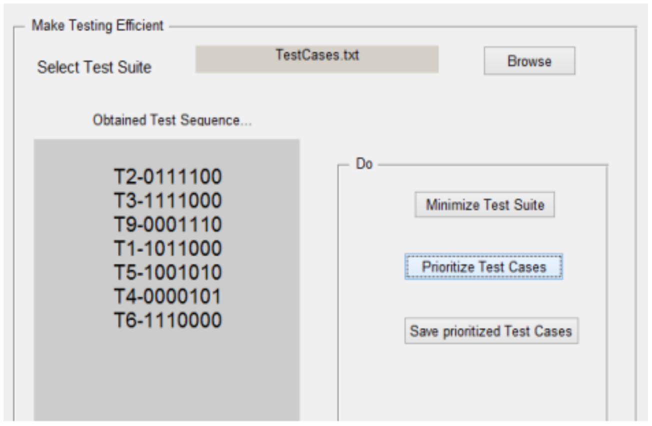
TABLE 2: PARAMETER WITH MINIMUM CASE TEST

Parameter	Value
Number of Test Cases	7
Fitness Function	Minimization
Generation	100
Test cost	0 to 1
Crossover	PMX

TABLE 3: NUMBER OF FAULT IN THE TEST CASE

Test	Case Fault			
	1	2	3	4
T7	X	X		
T8	X			X

Our greedy algorithm may select test case t1 first, test case t2 second, and test case t3 third. However, the optimal test case orderings in this case are t2, t3, t1 and t3, t2, t1. Despite this fact, as we shall show, our algorithm pro vides a useful benchmark against which to measure practical techniques, because we know that an optimal ordering could perform no worse than the ordering that we calculate. For brevity, in the rest of this paper, we refer to our technique that incorporates this algorithm as optimal prioritization.



```
T2-0111100
T3-1111000
T9-0001110
T1-1011000
T5-1001010
T4-0000101
T6-1110000
```

Fig.5. Prioritized Test Sequence

The output based on this random cost assignment is:-

- The obtained Test Sequence of this assignment is
  T2->T3->T9->T1->T5->T4->T6
- The Process cost driven from the genetic on initial cost assignment is Process cost = 6.979843
- The cost driven after implementation of optimized sequence is Test Cost = 1.2563

VII. CONCLUSION

We proposed an approach which combines the Minimization technique and prioritization techniques of Regression Testing. Genetic algorithm is best choice for prioritization as using this algorithm fairly large number of time we will get optimum solution. This approach may be used by the software practitioners to reduce the time and effort required for prioritization of test cases in the test suite. The proposed approach may lead to greater savings of time and effort in larger and complex projects as compared to smaller ones. Using GA approach, software practitioners can effectively select & prioritize test cases from a test suite, with minimum execution time. The algorithm is solved manually and is a step towards Test Automation. In future an automation tool is to be developed to implement the proposed algorithm which can solve large number of test cases in efficient time.

VIII. REFERENCES

[1] Chartchai Doungsa et. al., “An automatic test data generation fromUML state diagram using genetic algorithm”,http://eastwest.inf.brad.ac.uk/document/publication/Doungsa-ardSKIMA.pdf, Accessed on 25.10.2012.

[2] K.K. Aggarwal, and Y. Singh, “A book on software engineering”, New Age International (P) Ltd.; Publishers, 4835/24, Ansari Road,Daryaganj, New Delhi, 2001.

[3] Vishnu Raja.P ,MuraliBhaskaran.V. ―Improving the Performance of Genetic Algorithm by Reducing thePopulation Size‖, International Journal of Emerging Technology and Advanced Engineering , Volume 3, Issue 8, August 2013

[4] Md. ImrulKayes ,‖Test Case Prioritization for regression Testing Based on Fault Dependency‖, IEEE ,ICECT,2011, 3rd International Conference, Volume 5,Pages 48-52.

[5] HimanshuGhetia , ShantanuSantold, Vairamuthu S , ―Test case Prioritization based on testing requirementPriorities and fault dependency‖, International journal of Advanced Research in Computer Science and Software Engineering, Volume 4, Issue 6, June 2014

[6] A.P. Mathur, Foundations of Software Testing, New Delhi, Addison-Wesley Professional, 2008Examples:

[7] Huang C., Huang Y., Chang R., and Chen Y.,“Design and Analysis of Cost-Cognizant Test Case Prioritization Using Genetic Algorithm with Test History,” in Proceedings of the 34th Annual Computer Software and Applications Conference, Seoul, South Korea, pp. 413-418, 2010.

[8] Kaur A. and Goyal S., “A Bee ColonyOptimization Algorithm for Code Coverage TestSuite Prioritization,” International Journal of Engineering Science and Technology, vol. 3, no.4, pp. 2786-2795, 2011.

[9] grawal H., Horgan R., Krauser W., and LondonS., “Incremental Regression Testing,” in Proceedings of

International Conference on Software Maintenance, CSM-93, Montreal, Canada, pp. 348-357, 1993.

[10] R Walcott , ―Prioritizing regression test suites for time constrainted execution using genetic algorithm‖.[online] available at www.cs.virginia.edu/~krw7c/gaprioritization.pdf