Journal of Theoretical and Applied Information Technology
31st July 2014. Vol. 65 No.3

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

MULTI-FACTOR APPROACH FOR EFFECTIVE REGRESSION TESTING USING TEST CASE OPTIMIZATION

[1]S RAJU, [2]G V UMA

[1]Associate Professor, Department of Computer Science & Engg
Sri Venkateswara College of Engineering, Sriperumbudur, India - 602117
[2]Professor, Department of Information Science & Technology
Guindy College of Engineering, Anna University, Chennai, India - 600025
[1]Corresponding Author Email: srajuhere@gmail.com , gvuma@annauniv.edu

ABSTRACT

Regression testing intends to ensure that a software applications works as specified after changes have been made to it, is an important phase in software development lifecycle. Regression testing is the re-execution of some subset of test that has already been conducted. In regression testing, number of regression tests increases and it is impractical and inefficient to re execute every test for every application or function when change occurs. It is an expensive testing process used to detect regression faults. Regression testing has been used to support software-testing activities and assure acquiring an appropriate quality through several versions of a software product during its development and maintenance. Test suites can be large and conducting regression tests is tedious. Regression testing assures the quality of modified applications against unintended changes. The test case selection and prioritization is important in regression testing. Test case prioritization seeks to find an efficient ordering of test case execution for regression testing. Test case prioritization is used in regression testing, at the test suite level, with the goal of detecting faults as early as possible in the regression testing process, given a test suite inherited from previous versions of the system.

Keywords - *Regression Test, Test Case Prioritization, Priority Factors, Defect Density, Defect Removal Efficiency, Average Percentage of Fault Detected (APFD), Genetic Algorithm, Clustering.*

1. INTRODUCTION

Regression testing, which intends to ensure that a software program works as specified after changes have been made to it, is an important phase in software development lifecycle. Regression testing is the re-execution of some subset of test that has already been conducted. In regression testing as integration testing proceeds, number of regression tests increases and it is impractical and inefficient to re execute every test for every program function if once change occurs. It is an expensive testing process used to detect regression faults. Regression testing has been used to support software testing activities and assure acquiring an appropriate quality through several versions of a software product during its development and maintenance. Regression testing is an important and yet time consuming software development activity. It executes an existing test suite on a changed program to assure that the program is not adversely affected by unintended amendments. Test suites can be large and conducting regression tests is tedious. Regression testing assures the quality of modified service-oriented business applications against unintended changes. The test case prioritization is important in regression testing. It schedules the test cases in a regression test suite with a view to maximizing certain objectives which help reduce the time and cost required to maintain service-oriented business applications. Existing regression testing techniques for such applications focus on testing individual services or workflow programs. Test case prioritization seeks to find an efficient ordering of test case execution for regression testing. The most ideal ordering of test case execution is one that reveals faults earliest. Since the nature and location of actual faults are generally not known in advance, test case prioritization techniques have to rely on available surrogates for prioritization criteria. Test suite prioritization is a regression testing technique where test cases are ordered such that faults can be detected early in the test

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

execution cycle. This is useful because tests accumulate over multiple revisions and versions of the system and it is not feasible to execute all the tests in a limited amount of time. Test case prioritization is used in regression testing, at the test suite level, with the goal of detecting faults as early as possible in the regression testing process, given a test suite inherited from previous versions of the system. There are many techniques for prioritizing test cases based on various forms of information such as code coverage or modification history. In test case prioritization techniques two dimensions are considered. The first is granularity and the second dimension is the prioritization strategy. Over the lifetime of a large software product, the number of test cases could drastically increase as new versions of software are released. Because the cost of repeatedly retesting all test cases may be too high, software testers tend to remove redundant or trivial test cases to construct a reduced test suite for regression testing at a reasonable cost. After development and release, software undergo regress maintenance phase of ten to fifteen years. Modifications in software may be due to change in customer's requirements or change in technology or platform. This leads to release of numerous versions or editions of the existing software.

Regression testing is a process of executing the program to detect defects by retesting the modified portion or entire program. This can be performed by running the existing test suites or a new extended test suite against the modified code to determine whether the changes affect the entire program that worked properly prior to the changes. Adequate coverage will be a primary concern when conducting regression tests. The process of regression testing can be stated as follows. Let S be a program and S' be a modified version of program S; let T be a set of test cases for P then T' is selected from T that is subset of T for executing P', establishing T' correctness with respect to P'. Regression testing process consisted of steps that include Regression test selection problem, Coverage identification problem, Test suite execution problem and Test suite maintenance problem. Sometimes, the existing test suit may not be sufficient to test the modified code. In such case, an extended test suite is required to cover the defects created due to modifications. Modifications to the current version of the software can be an addition or deletion of new features in terms of modules or altering the existing features.

2 RELATED WORK

A handful of researches have been presented in the literature for the prioritization of regression testing test cases. Recently, utilizing artificial intelligence techniques like Greedy Algorithm and Genetic Algorithm (GA), in Prioritization has received a great deal of attention among researchers. A brief review of some recent researches is presented here.

Yogesh et al. [1], [2] have proposed an approach that variables were vital source of changes in the program and test cases should be prioritized according to the variables of any changed statement and variables computed from the variables of changed statements.

R.Kavitha et al. [3], [4] have proposed a prioritization technique to improve the rate of fault detection of severe faults for Regression testing. Here, two factors rate of fault detection and fault impact for prioritizing test cases are proposed. The results prove that the proposed prioritization technique was effective.

Ruchika et al. [5] have proposed both regression test selection and prioritization technique. They implemented their regression test selection technique and demonstrated that their technique was effective regarding selecting and prioritizing test cases. The proposed technique increases confidence in the correctness of the modified program.

A Kaur et al. [6],[7] proposed an algorithm to prioritize test cases using Genetic Algorithm. The genetic algorithm was introduced that will prioritize regression test suite within a time constrained environment on the basis of total fault coverage. The APFD has been calculated to evaluate the usefulness of the proposed algorithm.

Sanjukta Mohanty et al.[8] proposed a test case prioritization technique based on the factors such as code, requirements and model-based prioritization techniques and implement in CBSS. There was good coverage in terms of research in understanding the concepts of different code based techniques and behavior of components, interactions and compatibility of components.

Jayant et al. [9] have proposed a study on test case prioritization based on cost, time and process aspects. Prioritization concept increases the rate of

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

fault detection or code in time and cost constraints. They have concluded that prioritization of test case or a test suit has different aspects of fault detection.

S Raju and G V Uma [10] it was shown that the test case prioritization involves scheduling test cases in an order that increases the effectiveness in achieving some performance goals. One of the most important performance goals is the rate of fault detection. Test cases should run in an order that increases the possibility of fault detection and also that detects the most severe faults at the earliest in its testing life cycle. In this paper, we develop and validate requirement based system level test case prioritization scheme to reveal more severe faults at an earlier stage and to improve customer-perceived software quality using Genetic Algorithm (GA). For this, we propose a set of prioritization factors to design the proposed system. In our proposed technique, we refer to these factors as Prioritization Factors (PF). These factors may be concrete, such as test case length, code coverage, data flow, and fault proneness, or abstract, such as perceived code complexity and severity of faults, which prioritizes the system test cases based on the six factors: customer priority, changes in requirement, implementation complexity, completeness, traceability and fault impact. The goodness of these orderings was measured using an evaluation metric called APFD and PTR that will also be calculated.

S Raju and G V Uma, [11], Test case prioritization techniques have been shown to be beneficial for improving regression-testing activities. With prioritization, the rate of fault detection is improved, thus allowing testers to detect faults earlier in the system-testing phase. Most of the prioritization techniques to date have been code coverage-based. These techniques may treat all faults equally. Test case prioritization techniques schedule test cases for execution so that those with higher priority, according to some criterion are executed earlier than those with lower priority to meet some performance goal. In this paper, we introduce a cluster-based test case prioritization technique. By clustering test cases, based on their dynamic runtime behavior, we can reduce the required number of pair-wise comparisons significantly. We present a value-driven approach to system-level test case prioritization called the Prioritization of Requirements for Test. In this approach prioritization of test cases is based on four factors Rate of fault Detection, Requirements volatility, Fault impact and Implementation complexity. Our results show that this prioritization approach at the system level improves the rate of detection of severe faults.

S Raju and G V Uma, [12], Regression testing intends to ensure that a software applications works as specified after changes made to it during maintenance. It is an important phase in software development lifecycle. Regression testing is the re-execution of some subset of test cases that has already been executed. It is an expensive process used to detect defects due to regressions. Regression testing has been used to support software-testing activities and assure acquiring an appropriate quality through several versions of a software product during its development and maintenance. Regression testing assures the quality of modified applications. In this proposed work, a study and analysis of metrics related to test suite volume was undertaken. It was shown that the software under test needs more test cases after changes were made to it. A comparative analysis was performed for finding the change in test suite size before and after the regression test.

3 RESEARCH PROBLEM

The Proposed research work consisted of two different approaches for test case optimization. First approach uses genetic algorithm for test case prioritization. The second approach uses cluster approach for test case optimization. The first approach uses 7 factors and the second approach uses 4 factors that influences the successful implementation and execution of software projects. Values for these factors are determined using Goal-Question-Metrics (GQM) approach. The research work carried out here addresses the following research questions.

RQ1. What is the effect of adding new features and modifying existing features of the current release over the previous releases of software?
RQ2. Whether the existing Test Suit is good enough to test the modified version of the program?
RQ3. What is the effect of modification of software projects on the test suite volume size?
RQ4. How do the metrics (i) Defect Density (ii) Test Case Efficiency (iii) Average Percentage of Fault Detected vary before and

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

after regression testing?

4 TEST CASE SELECTION

Regression testing involves reusing test suites which have been created for earlier versions or releases of the software. By reusing these test cases, the costs of designing and creating test cases can be amortized across the lifetime of a system. The issue of retesting of software systems can be handled using a good test case prioritization technique. A prioritization technique schedules the test cases for execution so that the test cases with higher priority executed before lower priority. The objective of test case prioritization is to detect fault as early as possible.

An improved rate of fault detection during regression testing can let software engineers begin their debugging activities earlier than might otherwise be possible, speeding the release of the software. Test case prioritization techniques improve the cost-effectiveness of regression testing by ordering test cases such that those that are more important are run earlier in the testing process. Prioritization can provide earlier feedback to testers and management, and allow engineers to begin debugging earlier. It can also increase the probability that if testing ends prematurely, important test cases have been run.

Here, we will introduce a new regression test suite prioritization algorithm that uses genetic approach for prioritizing the test suite with the aim of maximizing the number of faults to be detected. We have conducted experiments with different types of applications and used the genetic algorithm approach to prioritize test cases The ordering was identified using test case priority

The proposed research work, computation of certain factors such as customer assigned priority of requirements, implementation complexity, changes in requirements, fault impact of requirements, completeness, traceability and Execution time are essential for prioritizing the test cases because they are used in the prioritization algorithm. These factors are derived using GQM Methodology. Using these factors a weight is obtained for each test case based on which the test cases are prioritized. Values for all the seven factors are obtained for each requirement during the design, analysis phase and evolve continually during the software development process as the project evolves.

Requirement factor value for each requirement i, Rfv_i is computed as follows:

$$Rfv_i = \frac{\sum_{j=1}^{7} factor_j}{7} \quad (1)$$

Significantly it represents the requirement factor value for requirement i, which is the mean of factor value. RFV is a measure of importance of testing a requirement and it is used in the computation of test case weight (TCW). With the total of n requirements, if test case t maps to i number of requirements then the test case weight Tcw_t is computed as follows.

$$Tcw_t = \left(\sum_{a=1}^{i} Rfv_a \Big/ \sum_{b=1}^{n} Rfv_b \right) * i/n \quad (2)$$

The test cases are sorted for execution based on the descending order of TCW, such that the test case with the highest TCW runs first. The Test Case Weights and RFV values are given as input to the GA for the prioritization of test cases. The test case with maximum fitness value will be elected as the high priority test case.

5 PROPOSED SYSTEM ARCHITECTURE

The proposed system uses Junit 3.8.1 Software Testing Tool and executed in Net Beans 7.1 IDE, Java SDK 1.6. The architecture of the proposed system is shown in figure 5.1 given below.

Journal of Theoretical and Applied Information Technology
31st July 2014. Vol. 65 No.3

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

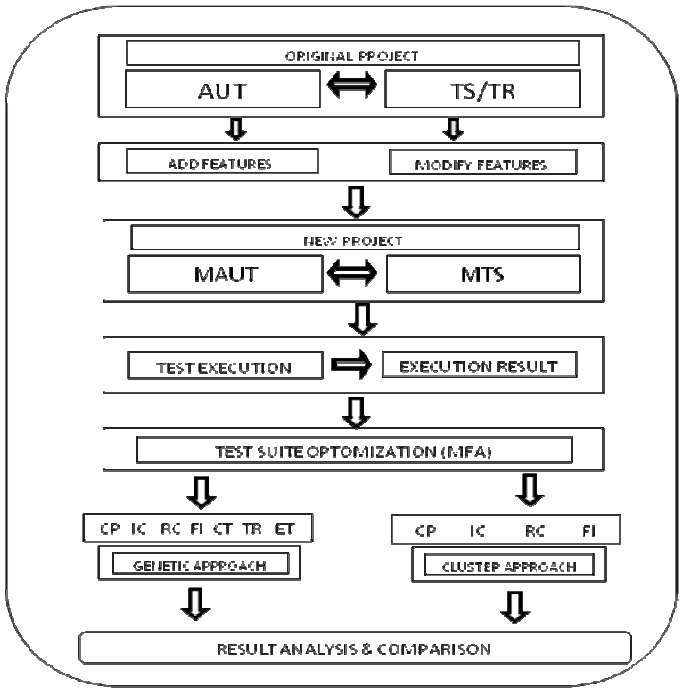

Figure 5.1 Proposed System Architecture

The proposed research work consider two categories of application programs. The first kind contains five programs that are small in size varies up to 1 KLOC and the second categories contain larger applications projects taken from Industries, whose size varies up to 30 KLOC, are shown in the table 5.1.

Table 5.1 Subject Programs

SL. NO	Problem /Project	Size (S) (loc/kloc)	No. of Modules (M)	No. of Defects Found (D)	Test Suite Size (N)
1	Triangle Classification	25	5	12	35
2	Square Root Problem	19	4	9	24
3	Electricity Bill Generation	155	13	20	96
4	Simple Calculator Program	250	18	38	126
5	Simple Editor Program	452	29	69	204
LARGER APPLICAIONS					
1	Payroll System	15	60	1012	1435
2	Infrastructure Mgt. System	21	64	1290	1524
3	Library System	8	45	629	1096
4	Project Mgt. System	25	75	2638	2926
5	Banking System	31	94	3869	4204

Journal of Theoretical and Applied Information Technology
31st July 2014. Vol. 65 No.3

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

6 TOOL SUPPORT AND EXPERIMENTS

These applications are taken to experiment the effectiveness of Testing after modifications and new additions. With industry data, we calculated few metrics - Defect Density per LOC and Test Case Efficiency, which are shown in the tables 6.1 for small programs and large projects.

Defect density is obtained by dividing number of defects covered by the program/project size and Test Case Efficiency is calculated as percentage of defect covered divided by the number of test cases. Since the proposed research work address the issue of effective regression testing, these projects are modified in two ways. Either a set of new features/modules are added or/and the existing modules are modified.

Table 6.1. Defect Density And Test Case Efficiency

SL. NO	Problem / Project	Test Suite Size (N)	No. of Defects Covered (D)	Defect Density per loc/kloc (D/S)	TC Efficiency (D/N)*100
1	Triangle Classification	35	12	0.48	25.71
2	Square Root Problem	24	9	0.47	41.67
3	Electricity Bill Generation	96	20	0.13	20.83
4	Simple Calculator Program	126	38	0.15	30.20
5	Simple Editor Program	204	69	0.15	33.82
LARGER PROJECTS					
1	Payroll System	1435	1012	67.47	70.52
2	Infrastructure Mgt. System	1524	1290	61.43	84.65
3	Library System	1096	629	78.63	57.39
4	Project Mgt. System	2926	2638	105.52	90.16
5	Banking System	4204	3869	124.81	92.03

The applications are tested and the results shows that the new faults are generated. Also to test these faults, new test cases are required hence the test suite is updated accordingly. The results after the modifications for small programs and large projects are summarized in the table 6.2.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Table 6.2. Effect Of Modifications - Small & Large Application Programs

SL. NO	Problem / Project	Size (S) (LOC/ KLOC)	No. of Modules (M)	No. of Defects Found (D)			Test Suite Size (N)		
				Old	New	Total	Old	New	Total
1	Triangle Classification	20	5	12	9	17	35	6	41
2	Square Root Problem	22	4	9	10	19	24	5	29
3	Electricity Bill Generation	195	15	20	9	29	96	10	106
4	Simple Calculator Program	290	20	38	7	45	126	9	135
5	Simple Editor Program	402	30	69	11	80	204	9	213
LARGE PROJECTS									
1	Payroll System	15.4	65	1012	57	1069	1435	46	1481
2	Infrastructure Mgt. System	21.3	67	1290	62	1352	1524	55	1579
3	Library System	8.5	51	629	24	653	1096	40	1136
4	Project Mgt. System	25.4	73	2638	48	2686	2926	47	2973
5	Banking System	30.6	90	3869	81	3950	4204	52	4256

After the regression testing, we have calculated the metrics - Defect Density per LOC and Test Case Efficiency, which are shown in the tables 6.3 respectively for small programs and large projects.

Table 6.3. Defect Density And Test Case Efficiency

Sl. NO	Problem / Project	Size (S) LOC	Test Suite Size (N)	No. of Defects Covered (D)	Defect Density per LOC (D/S)	TC Efficiency (D/N)*100
1	Triangle Classification	20	41	17	0.850	41.463
2	Square Root Problem	22	29	19	0.863	65.517
3	Electricity Bill Generation	195	106	29	0.149	27.358
4	Simple Calculator Program	290	135	45	0.155	33.333
5	Simple Editor Program	402	213	80	0.199	37.558

Journal of Theoretical and Applied Information Technology
31st July 2014. Vol. 65 No.3

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

LARGE APPLICATIONS						
1	Payroll System	15.4	1481	1069	69.416	72.181
2	Infrastructure Mgt. System	21.3	1579	1352	63.474	85.624
3	Library System	8.5	1136	653	76.824	57.482
4	Project Mgt. System	25.4	2973	2686	105.748	90.346
5	Banking System	30.6	4256	3950	129.085	92.810

Figures 6.1(a) and 6.1 (b) shows the effect of defect density before and after regression testing for small and larger applications/programs..

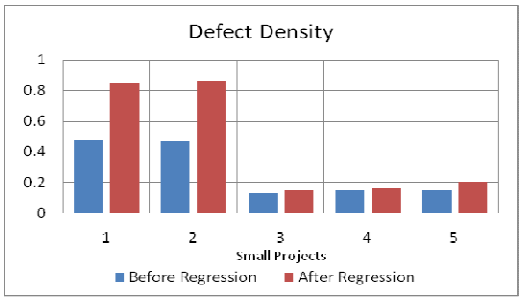

Figure 6.1 (A) Defect Density Before And After Regression Testing For Small Programs

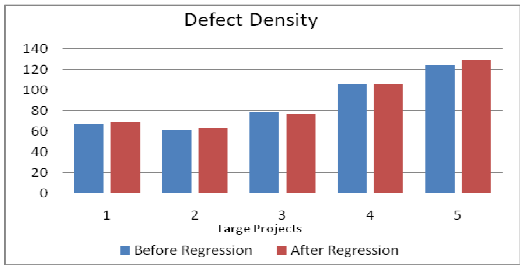

Figure 6.1 (B) Defect Density Before And After Regression Testing For Large Applications

Defect Removal Efficiency

Defect Removal Efficiency (DRE) is given by the formula,

DRE = (E/ E+D) x 100

where E = No. of Defects (Before Modifications) and D= No. of Defects Newly Introduced / Created (After Modifications).

DRE Metric computations after regression testing are summarized in Table 6.4 for small programs and larger applications after Regression Testing.

Table 6.4 Defect Removal Efficiency After Regression Testing

S. NO	Problem / Project	No. of Defects Found (D)		Defect Removal Efficiency	Test Suite Size (N) [(AM-BM)/BM] * 100		
		BM	AM	(BM/AM) * 100	BM	AM	TS Volume Increase (%)
1	Triangle Classification	12	17	70.59	35	41	17.1
2	Square Root Problem	9	19	47.37	24	29	20.83
3	Electricity Bill Generation	20	29	68.97	96	106	10.42

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

4	Simple Calculator Program	38	45	84.44	126	135	7.14
5	Simple Editor Program	69	80	86.25	204	213	4.41
6	Payroll System	1012	1069	94.67	1435	1481	3.21
7	Infrastructure Mgt. System	1290	1352	95.41	1524	1579	3.61
8	Library System	629	653	96.32	1096	1136	3.65
9	Project Mgt. System	2638	2686	98.21	2926	2973	1.61
10	Banking System	3869	3950	97.95	4204	4256	1.24

7 TEST CASE PRIORITIZATION USING GENETIC ALGORITHM

7.1. Genetic Algorithm

We know that each chromosome consists of genes in GA, a population P= (c1….cm) is formed from a set of chromosomes. The GA increases the population of chromosomes by continuously replacing one with another population and it based on fitness function assigned to each chromosome. The strong one go further and the weak chromosome eliminated generation by generation. Crossover and mutation these are two main concepts in genetic algorithm.

7.1.1. Selection

There is a selection pattern to find which are chosen for mating and it is based on fitness and capability of an individual to survive and reproduce in an environment. Selection generates the new one from the old one, thus starting a new generation. Each chromosome is examines in present generation to determine its fitness value. From all that we can say that fitness value of chromosome used to select for the next generation.

7.1.2. Crossover or Recombination

In crossover process, exchange of segments occurs between a pair of chromosomes and crossover is applied on a particular by switching one of its allele with another from another individual in the population. The resultant is very different from its parents. The code below suggests an implementation of individual using crossover:

$$Child1 = c * parent1 + (1 - c) * parent2$$

$$Child2 = (1 - c) * parent1 + c * parent2$$

7.1.3 Mutation

Mutation is a process wherein one allele of gene is randomly replaced by another to yield new structure. It alters an individual in the population. It can regenerate all or a single allele in the selected individual. To maintain integrity, operations must be secure or the type of information an allele holds should be taken into consideration. Mutation must be aware of binary operations, or it must be able to deal with missing values. A simple piece of code is mentioned below.

$child = generateNewChild( );$

7.1.4 Algorithm

The optimization problems are solved by GA's recombination and replacement operators, where recombination is key operator and frequently used, whereas, replacement is optional and applied for solving optimization of problem. Here, the initial population is automatically generated and the evaluation of the set of candidate solution has been done with the help of genetic algorithm. Here, test case weight (TCW) is used as the stopping criteria. The steps in the algorithm are given below. The optimal solution is searched in GA on the basis of desired population which further can be replaced with the new set of population. Depend upon the problem, the generation and initialization of test cases (population) is done. Requirement factor value (RFV) and test case weight (TCW) are chosen as the fitness criterion. Henceforth, this fitness function will help in selecting suitable population for problem. Further, the genetic operations are performed. In the beginning, crossover recombines the two individual. Then mutation randomly swaps the individuals. Thirdly, the redundant individuals are removed. Finally, the solution is checked for optimization. If solution is not optimized, then, the new population is reproduced and genetic operators are applied.

7.1.5 Performance of the Prioritization algorithm

To analyze the performance of the prioritization technique used in this research, the optimized test suit is considered for assessing the effectiveness of the sequence of the test cases. Effectiveness will be measured by the rate of faults detected. Consider a sample set of data for a

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

problem with 'n=8' number of requirements, size is 19 LOC and number of defects found is 9 using 11 test cases. RFV values are calculated using Goal-Question-Metrics (GQM) approach for the sample program and given in table 7.1. The method of calculating Test Case Weights (TCW) are shown in the table 7.2.

Table 7.1 Requirement Factor Values

Req.	CP	IC	RC	FI	CT	ET	RFV
1	1	0	1	1	6	2	1.571
2	3	0	1	1	7	3	2.143
3	2	0	1	1	5	1	1.429
4	4	1	2	2	8	4	3.000
5	8	1	2	2	9	2	3.429
6	3	0	1	1	5	2	1.714
7	2	0	0	1	6	2	1.571
8	6	1	1	2	9	2	3.000

Table 7.2 Test Case Weight (Tcw)

TC	Req. Mapped	RFV	i	i/n	NU	TCW	Priority
1	1	1.571	1	0.125	1.571	0.234	10
2	1, 2, 3	1.571, 2.143, 1.429	3	0.375	5.143	0.731	1
3	1, 2	1.571, 2.143	2	0.250	3.714	0.507	4
4	3, 4	1.429, 3.000	2	0.250	4.429	0.557	3
5	1, 4	1.571, 3.000	2	0.250	4.571	0.567	2
6	1, 3	1.571, 1.429	2	0.250	3.000	0.458	6
7	2	2.143	1	0.125	2.143	0.274	9
8	1, 7	1.571, 1.571	2	0.250	3.142	0.468	5
9	3	1.429	1	0.125	1.429	0.224	11
10	4	3.000	1	0.125	3.000	0.333	8
11	5	3.429	1	0.125	3.429	0.363	7

TCW = (sum/NU)+(i/n)

$$Sum = \sum_{b=1}^{n} RFVb$$

$$NU = \sum_{a=1}^{i} Rfv\ a$$

Journal of Theoretical and Applied Information Technology
31st July 2014. Vol. 65 No.3

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

7.1.6 Average Percentage of Fault Detected (APFD) Metric

Calculation of APFD Metric is shown in the table 7.3 and the performance is shown in the figure 7.1 as graph, proves that there is an improvement after regression testing.

Table 7.3 Test Case Execution Sequence

TC Defect	TC1	TC2	TC3	TC4	TC5	TC6	TC7	TC8	TC9	TC10	TC11
F1		X	X					X			
F2								X	X		
F3				X					X		
F4				X	X						
F5					X	X	X				
F6					X						
F7				X				X			
F8					X					X	
F9											X

To quantify the goal of increasing a subset of the test suite's rate of fault detection, we use a metric called APFD that measures the rate of fault detection per percentage of test suite execution.

The APFD is calculated by taking the weighted average of the number of faults detected during the run of the test suite. APFD can be calculated as follows:

$$Apfd = 1 - \frac{(Tf_1 + Tf_2 + ... + Tf_m)}{nm} + \frac{1}{2n}$$

Where,
n is the no. of test cases, Tf_1, Tf_2, Tf_3,... are the position of first test that exposes the fault.
m is the no. of faults.

APFD Metric Value Before Regression Testing is given by

$$Apfd(T,P) = 1 - \frac{(1+5+3+3+2+2+2+2+7)}{11*9} + \frac{1}{2*11}$$

$$= 0.7273 + 0.0454$$

$$= 0.7727$$

APFD Metric Value after Regression Testing is given by

$$Apfd(T,P) = 1 - \frac{(2+8+4+4+5+5+4+5+11)}{11*9} + \frac{1}{2*11}$$

$$= 0.5152 + 0.0454 \text{ \% of fault Detected}$$

$$= 0.5606$$

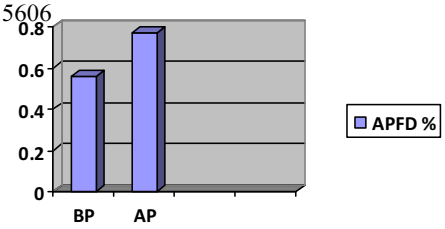

Figure 7.1 APFD Metric

7.2 Results and Discussion

Test Case Efficiency after optimization has shown to improved and Test Suite size also reduced after the optimization. These results are shown in the table 7.4. The proposed Multi factor

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Approach for Effective Regression Testing using Test Case Optimization system shows an improvement over the existing approaches. Figure 7.2 shows the Test Case Efficiency before and after regression and after optimization of test cases. The TC efficiency improves after optimization and modifications.

Table 7.4 Test Case Efficiency After Optimization For Small & Large Projects

Sl. NO	Problem / Project	TC Efficiency (D/N)*100		
		Before Regression	After Regression	After Optimization
1	Triangle Classification	25.71	41.46	68.00
2	Square Root Problem	41.67	65.52	86.36
3	Electricity Bill Generation	20.83	27.36	60.42
4	Simple Calculator Program	30.16	33.33	46.88
5	Simple Editor Program	33.82	37.56	48.19
6	Payroll System	70.52	72.18	87.27
7	Infrastructure Mgt. System	84.65	85.62	91.47
8	Library System	57.39	57.48	76.29
9	Project Mgt. System	90.16	90.35	96.07
10	Banking System	92.03	92.81	95.18

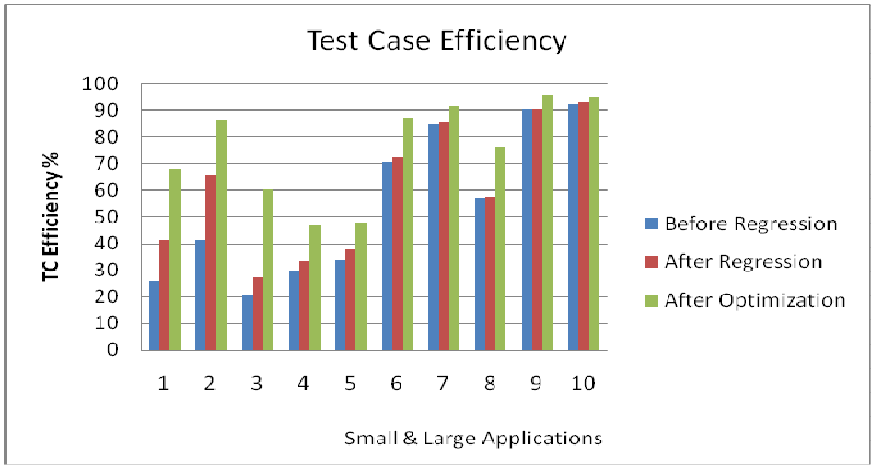

Figure 7.2 Test Case Efficiency

Journal of Theoretical and Applied Information Technology
31st July 2014. Vol. 65 No.3

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

8 CLUSTER BASED APPROACH

8.1 Priority Factors

In the second part of research work, only four factors such as rate of fault detection, requirements volatility, fault impact, and implementation complexity for Prioritization of Test Cases.

The average number of faults per time unit detected by a test case is called rate of fault detection. The rate of fault detection of test case i have been calculated using the number of faults detected and the time taken to find out those faults for each test case in a test suite.

$$RFTi = \frac{No.\ of\ \ faults}{time} * 10$$

This factor is converted into 1 to 10 point scale. The test case that have higher rate of fault detection will be given priority over the other test cases.

Requirements volatility (RV) is based on the number of times a requirement has been changed during the development cycle. If a requirement has changed more than certain number of times, the volatility values for all requirements are quantified on a 10-point scale. Changing requirements result in re-design, the addition or deletion of existing functions, and often an increase in the fault density in the program. Both fault impact and implementation complexity are calculated as described in part 1 of this research work.

8.2 Methodology

The diagram shown in the Figure 8.2 describe this Proposed Work. We use clustering concept for combining test cases. Instead of prioritizing individual test cases, clusters of test cases are prioritized using techniques such as Prioritization of Requirements for Test algorithm. Within each prioritized cluster, an optimal test case ordering can be achieved. The fault detection capability of each test case is used for creating the clusters of test cases.

For Each Cluster, C_k
For every Test Case, t
(i) Calculate PFV_i using

$$PFV_i = \sum_{j=1}^{4} \left(FactorValue_{ij} * FactorWeight_j \right)$$

(ii) Calculate Weighted Priority (WP) of test cases.

$$WP_j = \left[\frac{\sum_{x=1}^{i} PFV_x}{\sum_{y=1}^{n} PFV_y} \right] * \left(\frac{i}{n} \right)$$

Order the test cases based on WP values in each cluster.

Figure 8.3 Pseudo-code for Cluster Formation

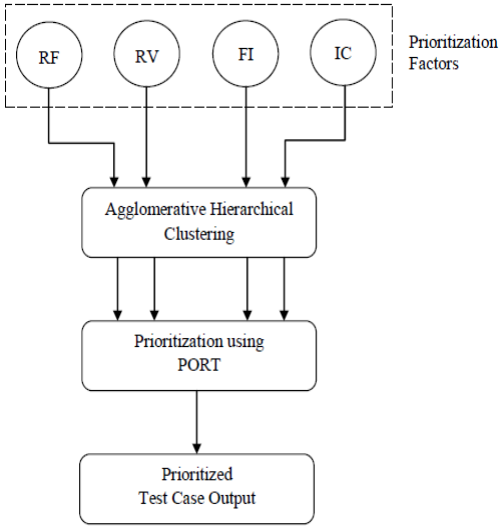

Figure 8.2 Clustering Process

8.3 Test Case Clustering

When considering test case prioritization, the ideal clustering criterion would be the similarity between the faults detected by each test case. However, this information is inherently unavailable before the testing task is finished. Therefore, it is necessary to find a surrogate for this, in the same way as existing coverage based prioritization techniques turn to surrogates for fault-detection capabilities. Here, we use a simple hierarchical clustering technique [16]. Its pseudo-code is shown in the figure 8.3.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

8.4 Test Case Prioritization

The resulting dendrogram is a tree structure that represents the arrangement of clusters. The rate of fault detection of each test case is used for forming clusters of test cases. These clusters are then considered for prioritization using PORT algorithm. The high level algorithm shown in the figure 8.4 describes the process of prioritization of test cases.

8.5 Experiments and Results

The same set of sample programs are considered here and the results shows that the results obtained based on applying genetic approach is better than that of applying cluster approach. The results shows that the proposed research technique described in this thesis are better than the work carried out by others. Results summarized in the table 8.5 shows that the genetic approach is better than cluster approach of test case optimization.

Table 8.5 Test Case Efficiency After Prioritization Using Cluster Approach

S. No	Problem / Project	No. of Defects (D)		Test Suite Size (N)		No. of clusters	Optimized Test Suite Size (N)	TC Efficiency (D/N)*100 (AP)	
		BM	AM	BM	AM			AM	AP
1	Triangle Classification	12	17	35	41	4	38	41.463	44.73
2	Square Root Problem	9	19	24	29	5	26	65.517	73.08
3	Electricity Bill Generation	20	29	96	106	7	99	27.358	30.21
4	Simple Calculator Program	38	45	126	135	11	130	33.333	34.62
5	Simple Editor Program	69	80	204	213	14	186	37.558	43.01

9. Performance Analysis

9.1 The APFD Metrics

The APFD metric calculation is shown below in the Table 9.1 below for different projects before and after prioritization.

Journal of Theoretical and Applied Information Technology
31st July 2014. Vol. 65 No.3

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Table 8.1 Apfd Metric

Problem / Project	APFD Metric (%)	
	Before Prioritization	After Prioritization
Triangle Classification	42	74
Square Root Problem	40	77
Electricity Bill	56	82
Simple Calculator	54	86
Simple Editor	43	80
Payroll System	56	86
Infrastructure Mgt. System	55	87
Library System	54	88
Project Mgt. System	53	82
Banking System	52	86

Figure 9.1 given below shows the APFD metric our projects.

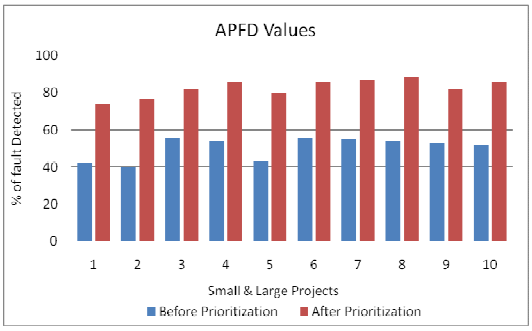

Fig. 9.1 The Apfd Metric

The APFD metric values of our project are compared with others work and shown to be better in most of the cases. It is shown Table 9.2.

Table 9.2. APFD Metric Comparison

Approaches	APFD Metric	
	Before Prioritization	After Prioritization
Arvindar Gaur et all	40 %	84 %
Our Proposed GA Approach	56%	86%

9. CONCLUSION AND FUTURE WORK

In this research work, the regression testing based test suite prioritization technique was illustrated with industry projects. New optimization techniques has been proposed for optimizing test cases to improve the rate of fault detection. The research work reported here recognizes and assesses the challenges coupled with regression testing test case prioritization. The proposed methods uses multiple and most efficient factors to prioritize the test cases. These factors identifies the important test cases in the project. The APFD metric was used to evaluate the effectiveness of the proposed prioritization technique. Also it is shown experimentally that the numbers of test cases run to reveal the defects is less in case of proposed prioritized execution of test cases. Based on the various performance measures and metrics obtained, it is proved that the proposed Multi Factor Approach for Effective Regression Testing using Test Case Optimization system is effectively prioritizing the regression test cases.

REFERENCES

[1] Yogesh Singh, Arvinder Kaur and Bharti Suri, "Empirical Validation of Variable based Test Case Prioritization/Selection Technique," International Journal of Digital Content Technology and its Applications, Vol.3, No. 3, pp.116-123, Sep 2009.

[2] Yogesh Singh, Arvinder Kaur and Bharti Suri, "A Hybrid Approach for Regression Testing in Interprocedural Program," Journal of Information Processing Systems, Vol.6, No.1, pp.21-32, Mar 2010.

[3] R. Kavitha and N. Sureshkumar, "Test Case Prioritization for Regression Testing based on Severity of Fault," International Journal on

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Computer Science and Engineering, Vol. 2, No. 5, pp.1462-1466, 2010.

[4] R. Kavitha and N. Suresh Kumar, "Model Based Test Case Prioritization for Testing Component Dependency in CBSD Using UML Sequence Diagram," International Journal of Advanced Computer Science and Applications, Vol. 1, No. 6, pp.108-113, Dec 2010.

[5] Ruchika Malhotra, Arvinder Kaur and Yogesh Singh, "A Regression Test Selection and Prioritization Technique," Journal of Information Processing Systems, Vol.6, No.2, pp.235-252, Jun 2010.

[6] Arvinder Kaur and Shivangi Goyal, "A Bee Colony Optimization Algorithm For Code Coverage Test Suite Prioritization," International Journal of Engineering Science and Technology (IJEST),Vol. 3, No. 4 , pp.2786-2795,Apr 2011.

[7] Arvinder Kaur and Shubhra Goyal, "A Genetic Algorithm for Fault based Regression Test Case Prioritization," International Journal of Computer Applications, Vol. 32, No.8, pp.975-8887, Oct 2011.

[8] Sanjukta Mohanty, Arup Abhinna Acharya and Durga Prasad Mohapatra, "A Survey On Model Based Test Case Prioritization," International Journal of Computer Science and Information Technologies, Vol. 2,No.3,pp.1042-1047, 2011.

[9] K.P. Jayant and Ajay Rana, "Prioritization Based Test Case Generation In Regression Testing," International Journal of Advances in Engineering Research (IJAER), Vol.1, No.5, May 2011.

[10] Raju, S. and Uma G V, “Factors Oriented Test Case Prioritization Technique in Regression Testing using Genetic Algorithm”, European Journal of Scientific Research (EJSR), ISSN 1450-216X Vol.74 No.3 (2012), pp. 389-402, © EuroJournals Publishing, Inc. 2012.

[11] Raju, S. and Uma G V, “An Efficient method to Achieve Effective Test Case Prioritization in Regression testing using Prioritization Factors", Asian Journal of Information technology (AJIT), ISSN: 1682-3915, Vol. 11 No.5 (2012), pp. 169-180, © Medwell Journals 2012.

[12] Raju S and G V Uma, "Measurement and Analysis of Test Suite Volume Metrics for Regression Testing", International National Journal of Engineering Research and Applications (IJERA), ISSN: 2248-9622, Vol. 4, Issue 1 (2014), pp. 11-20.