RESEARCH ARTICLE OPEN ACCESS

Measurement and Analysis of Test Suite Volume Metrics for Regression Testing

S Raju[1] and G V Uma[2]

[1]Associate Professor, Department of Computer Science & Engineering, Sri Venkateswara College of Engineering, Sriperumbudur, Tamilnadu, India – 602 117
[2]Professor, Department of Information Science & Technology, College of Engineering Guindy, Anna University Chennai Tamilnadu, India – 600 025

Abstract
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.

Keyword – Regression Testing, Test Suite Volume, Defect Density, Defect Analysis, Defect Removal Efficiency

I. INTRODUCTION

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.

Constructing extended test suite to test the new version of the software needs more careful effort of the testers. Test suite volume obviously grows proportionately with number of modifications introduced. Addition of randomly generated test cases has been shown to be effective. Also combinatorial based approach for adding test cases was also effective. In this work, two different kinds of applications are considered for measuring the test metrics. First category of software applications are small in size, up to 1 KLOC and the second category of software applications are larger in size, varying 5 to 30 KLOC.

II. RELATED WORKS

The literature survey revealed that many researchers have attempted to study the metrics related to software regression testing and test suite size. A brief review of some recent research on this area is presented here. The objective of regression testing is to have the highest likelihood of finding the defects yet-to-detect with a minimum amount of time and effort. This measurement help us to manage and control the software testing process.

Kan and Konda classified test metrics into three categories: product metrics, project metrics and process metrics. The test metrics can be used to measure and improve quality of test process and/or software product. Test metrics are a subset of

software metrics - product metrics, process metrics [1][2].

Gregg Rothermel presented various methodologies for improving regression testing processes. The cost-effectiveness of these methodologies have been shown to vary with characteristics of regression test suites. One such characteristic involves the way in which test inputs are composed into test cases within a test suite. This article reports the results of controlled experiments examining the effects of two factors in test suite composition---test suite granularity and test input grouping---on the costs and benefits of several regression-testing-related methodologies: retest-all, regression test selection, test suite reduction, and test case prioritization. The results exposed the essential tradeoffs affecting the relationship between test suite design and regression testing cost-effectiveness, with several implications for practice [3].

Mrinal Kanti Debbarma presented that the Software metrics was applied to evaluate and assure software code quality. It requires a model to convert internal quality attributes to code reliability. High degree of complexity in a component (function, subroutine, object, class etc.) is bad in comparison to a low degree of complexity in a component. Various internal codes attribute which can be used to indirectly assess code quality. In this paper, they analyzed the software complexity measures for regression testing which enables the tester/developer to reduce software development cost and improve testing efficacy and software code quality. This analysis was based on a static analysis and different approaches presented in the software engineering literature [4].

Ruchika 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 [5].

R Kavitha 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[6].

Roya Alavi and Shahriar Lotfi presented a software system testing methodology that includes a large set of test cases. Test selection helps to reduce this cost by selecting a small subset of tests that are likely to reveal faults. The test selection helps to reduce cost by selecting a small subset of tests that find to faults. The aim is to find the maximum faults of program using minimum number of test instances[7].

Pakinam N. Boghdady explain that the software testing immensely depends on three main phases: test case generation, test execution, and test evaluation. Test case generation is the core of any testing process; however, those generated test cases still require test data to be executed which makes the test data generation not less important than the test case generation. This kept the researchers during the past decade occupied with automating those processes which played a tremendous role in reducing the time and effort spent during the testing process. This paper explores different approaches that had emerged during the past decade regarding the generation of test cases and test data from different models as an emerging type of model based testing. Unified Modeling Language UML models took the greatest share from among those models[8].

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

III. PROJECTS AND RELATED DATA

The proposed research work consisted of many modules. Our first module consisted of modifications to existing features and addition of new modules manually.. Each applications considered separately for identifying the segments where the proposed changes are to be made. In effect, the size of the applications projects will be increased. Only in case of small programs, the code size may or may not increase. Second module consisted of running the existing test suites against these new versions of the application projects. For re-testing, the Junit test tool is used under Net Beans IDE with Java JDK.

The test results are then examined for their completeness of test execution. If any test result indicates that test process is not completed, then we need to add new test cases to the existing test suite. For adding new test cases, we either follow random approach or combinatorics approach. Details of the small projects used in the proposed research work such as project size, test suite size, defects count are shown in the table 1 and that of large projects are shown in the table 2.

The metrics such defect density, Test Case Efficiency, Test Suite volume increase are calculated before regression testing and after the modifications and after regression testing.

Table 1 Small Programs & other details

SL. NO	Problem / Project	Size (LOC) (S)	No. of Modules / Functions (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

Table 2 Large Size Projects & other details

SL. NO	Problem / Project	Size (LOC) (S)	No. of Modules / Functions (M)	No. of Defects Found (D)	Test Suite Size (N)
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

IV. RESEARCH OBJECTIVES

The proposed research work address the following issues in detail. For answering these questions, regression testing is performed and the results are presented in table format as well as in the graphical format in the following sections.

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?

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. When an existing software projects are modified to incorporate the changes in user requirements, the code size increases proportionately. Also when we try to add new modules for adding new functionality, the code size and number of modules increases. These applications are taken to experiment the effectiveness of Testing after modifications and new additions. With industry data, we have calculated the test metrics - Defect Density per LOC or KLOC and Test Case Efficiency before performing regression testing. These metrics are shown in the table 3 and table 4 respectively for small and larger size 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 addresses 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 3 Defect Density and Test Case Efficiency for small programs

SL. NO	Problem / Project	Test Suite Size (N)	No. of Defects Covered (D)	Defect Density per LOC (D/S)	TC Efficiency (D/N)*100
1	Triangle Classification	35	12	0.480	25.714
2	Square Root Problem	24	9	0.474	41.667
3	Electricity Bill Generation	96	20	0.129	20.833
4	Simple Calculator Program	126	38	0.152	30.159
5	Simple Editor Program	204	69	0.153	33.823

Figure 1 shows the defect density before regression test for smaller size programs.

The defect density for each projects/programs is calculated by using the formula

$$Defect\ Density = \frac{\text{Number of defect covered}}{\text{Project Size (LOC or KLOC)}}$$

Test Case Efficiency is calculated using the formula

$$Test\ Case\ Efficiency = \frac{\text{Number of defect covered}}{\text{Number of test cases}} \times 100$$

It is obvious that when we add new set of functionalities, the code size and number of modules always increases. These applications are considered for conducting re-test so as to measure the effectiveness of Testing after modifications and new additions.

Figure 2 shows the test case efficiency for smaller size projects.

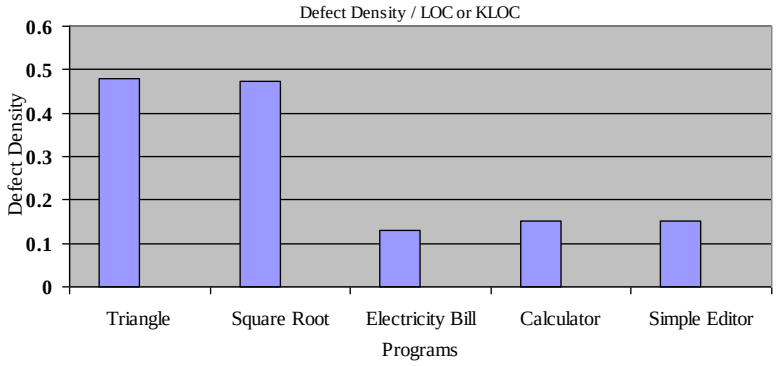
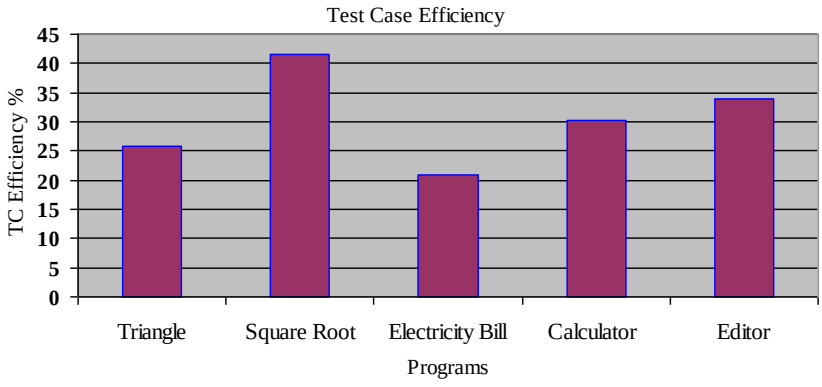


Fig. 1 Defect Density before Regression Testing for smaller size programs



\

Fig. 2 Defect Density before Regression Testing for Larger size programs

Table 4 Defect Density and Test Case Efficiency for Larger programs

SL. NO	Problem / Project	Test Suite Size (N)	No. of Defects Covered (D)	Defect Density per KLOC (D/S)	TC Efficiency (D/N)*100
1	Payroll System	1435	1012	67.467	70.523
2	Infrastructure Mgt. System	1524	1290	61.429	84.645
3	Library System	1096	629	78.625	57.391
4	Project Mgt. System	2926	2638	105.520	90.157
5	Banking System	4204	3869	124.806	92.031

Figure 3 shows the defect density before regression for larger size programs.

Figure 4 shows the test case efficiency for smaller size projects.

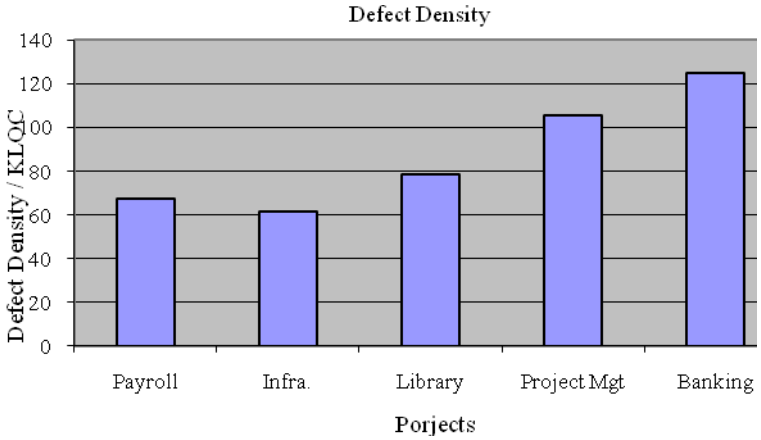


Fig. 3 Defect Density before Regression Testing for Larger Projects

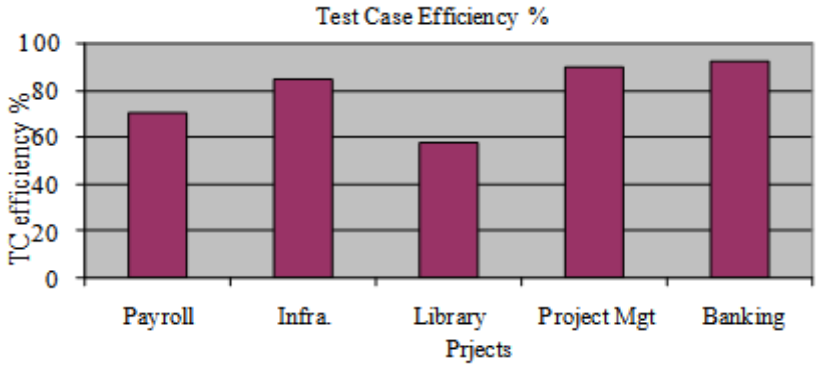


Fig. 4 Test Case Efficiency before Regression Testing for Larger Projects

V. REGRESSION TESTING OF APPLICATIONS

Before performing the regression testing we added new features and also modified existing features. Due to this code size of the projects/programs have increased. Consequent to this, the test suite is assed with more number of test cases so as to run the programs /projects against this extended test suite. Table 5 shows the details of small programs after the regression testing. It is observed that there is increase in test suite volume

and defect counts. Figure 5 given below shows that the number of defects increases when the programs are modified due to change in user requirements.

Table 5 Effect of Modifications for Small Programs

SL. NO	Problem / Project	Size (S) (LOC)	No. of Modules	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	195	15	20	9	29	96	10	106
4	Simple Calculator	290	20	38	7	45	126	9	135
5	Simple Editor Program	402	30	69	11	80	204	9	213

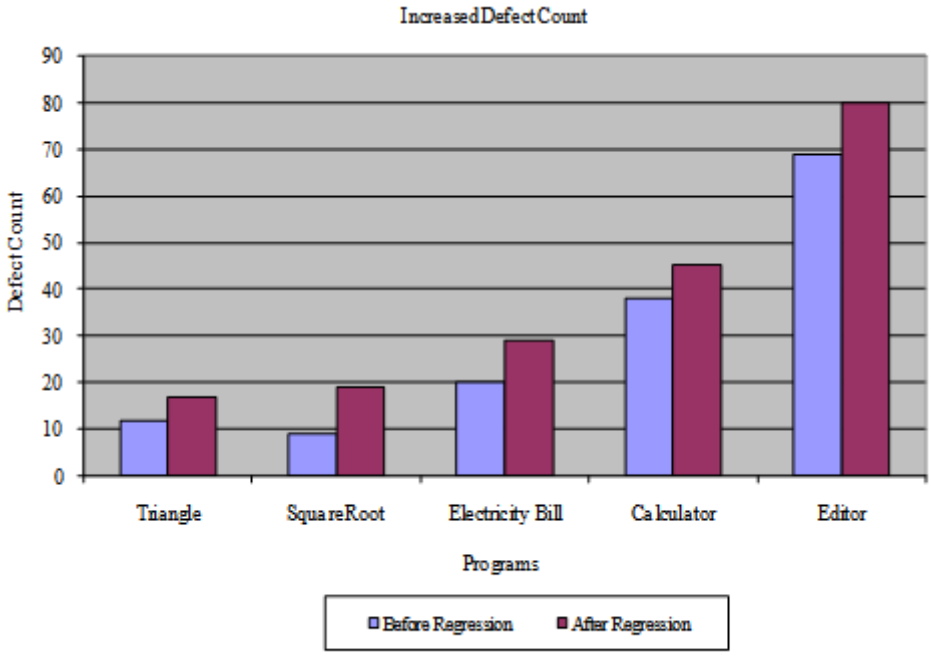


Fig 5 Number of defects before and after Regression Testing for Small Programs

The test suite volume increases for covering these additional defects due to modifications as shown in the figure 6 below.

Similarly, when we modify the larger projects to incorporate changes in user requirements, number of defects increases. Hence the original test suite should be added with more test cases to find these defects. This is shown in the table 6 given below.

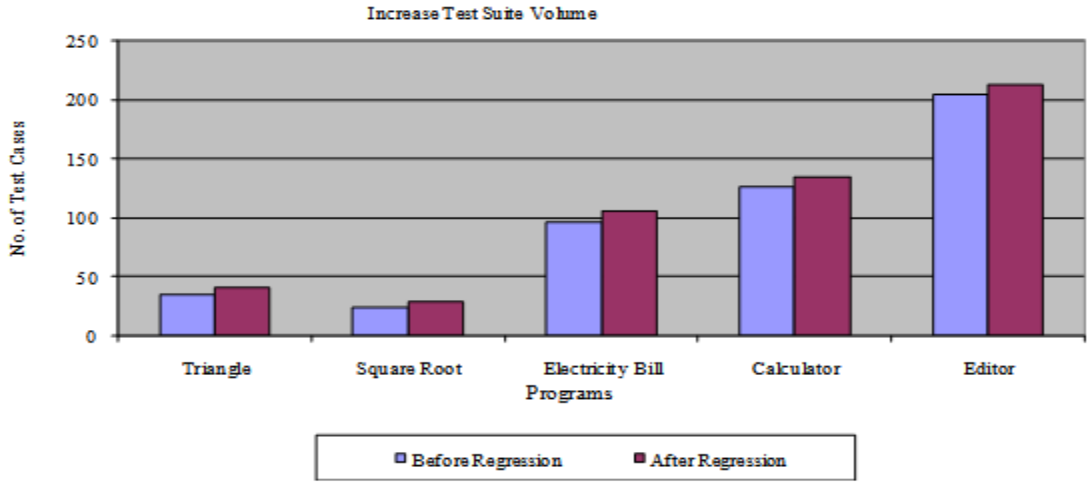


Fig 6 Increase of Test Suite Volume after Regression Testing for Large Programs

Table 6 Effect of Modifications for Larger Projects

Sl. NO	Problem / Project	Size (S) (*KLOC*)	No. of Module	No. of Defects Found (D)			Test Suite Size (N)		
				Old	New	Total	Old	New	Total
1	Payroll System	15.4	65	1012	57	1069	1435	46	1481
2	Infrastructure Mgt. Sys.	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

Figure 7 given below shows that the number of defects increases when the programs are modified due to change in user requirements in case of larger applications.

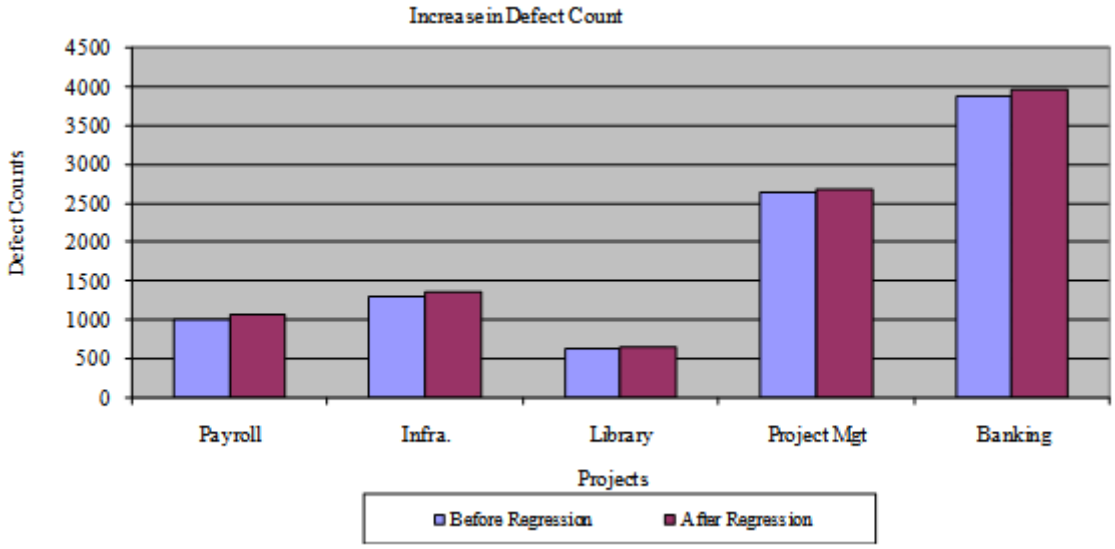


Fig 7 No. of defects before and after Regression Testing for Larger Programs

The test suite volume increases for covering these additional defects due to modifications as shown in the figure 8 below. When we compare the metrics defect density and test case efficiency of the test suites before and after the regression testing, both increased to some extent for most of the programs/projects.

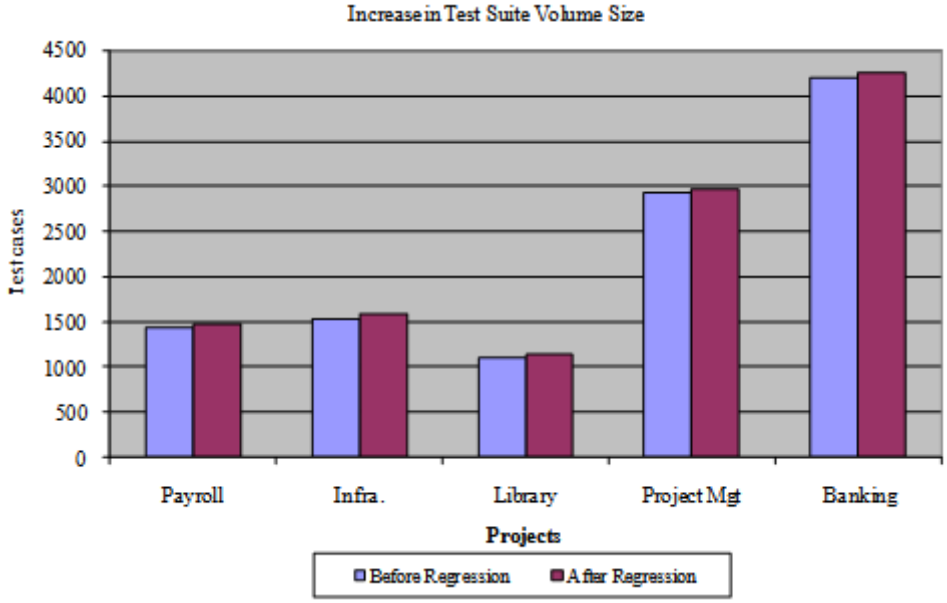


Fig 8 Increase of Test Suite Volume after Regression Testing for Larger Projects

This can be seen in the table 7 and graphically shown in the Figures 9(a) and Figure 9(b) below. Figures 10(a) and 10(b) shows the Test Case Efficiency before and after the Regression Testing.

Table 7 Defect Density and Test Case Efficiency after and before the Regression Testing

SL. NO	Problem / Project	Defect Density		Test Case Efficiency	
		BR	AR	BR	AR
1	Triangle Classification	0.480	0.850	25.714	41.463
2	Square Root Problem	0.474	0.863	41.667	65.517
3	Electricity Bill Generation	0.129	0.149	20.833	27.358
4	Simple Calculator Program	0.152	0.155	30.159	33.333
5	Simple Editor Program	0.153	0.199	33.823	37.558
6	Payroll System	67.467	69.416	70.523	72.181
7	Infrastructure Mgt. System	61.429	63.474	84.645	85.624
8	Library System	78.625	76.824	57.391	57.482
9	Project Mgt. System	105.520	105.748	90.157	90.346
10	Banking System	124.806	129.085	92.031	92.810

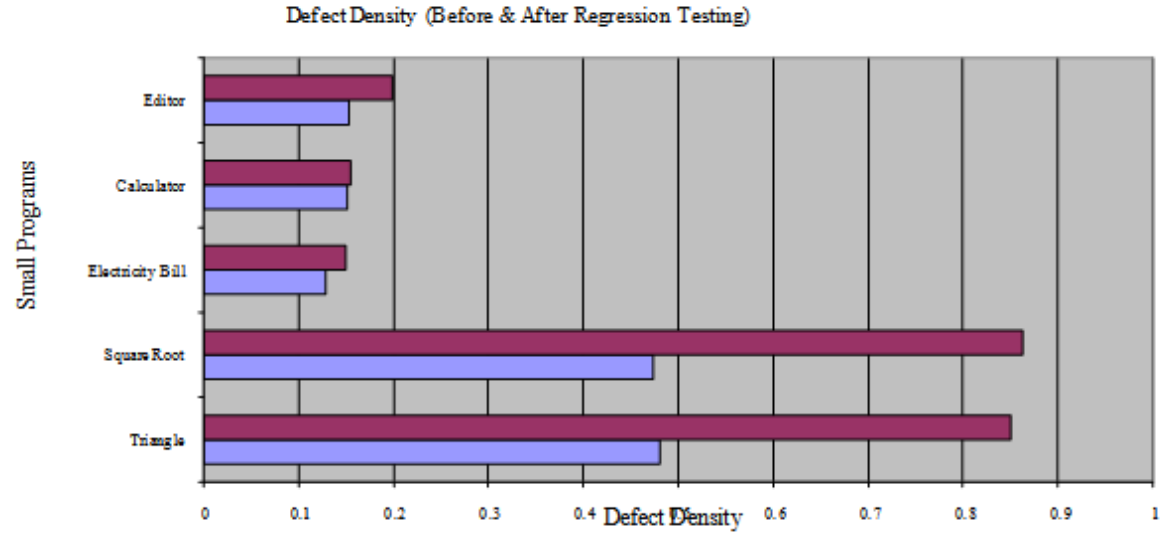


Fig 9(a) Defect Density for small Programs

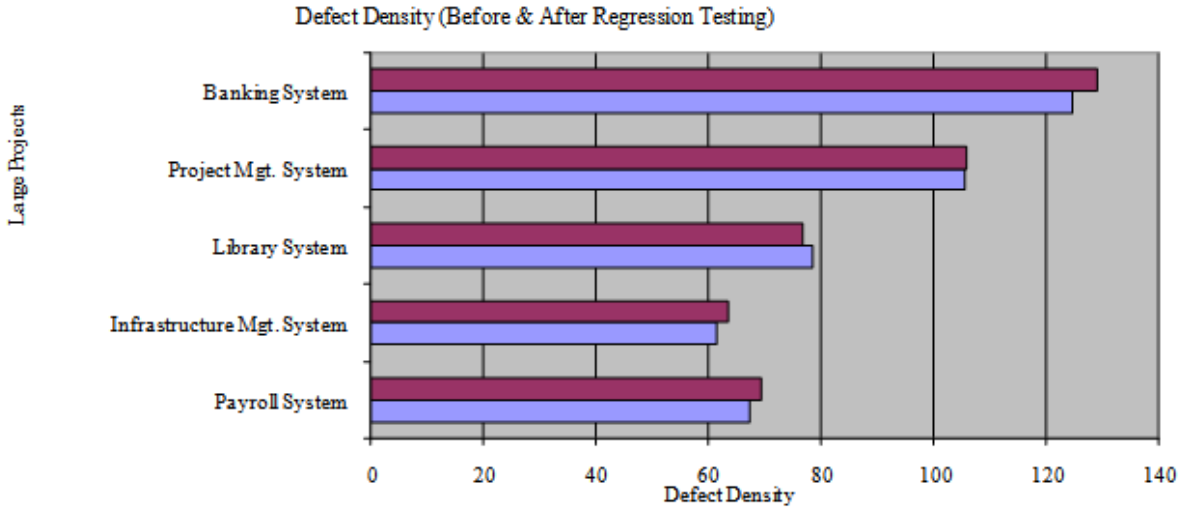


Fig 9(b) Defect Density for Large Programs

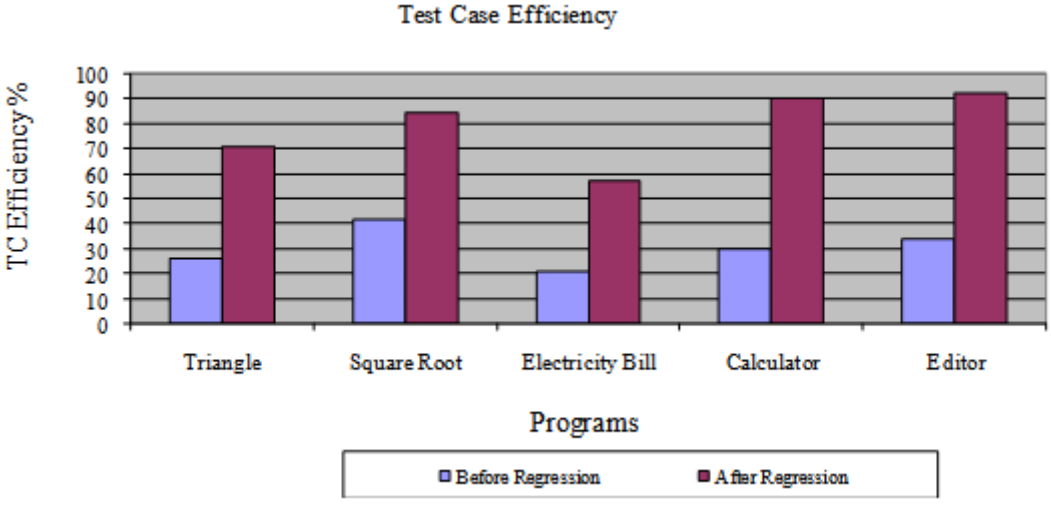


Fig 10(a) Test Case efficiency for small Programs

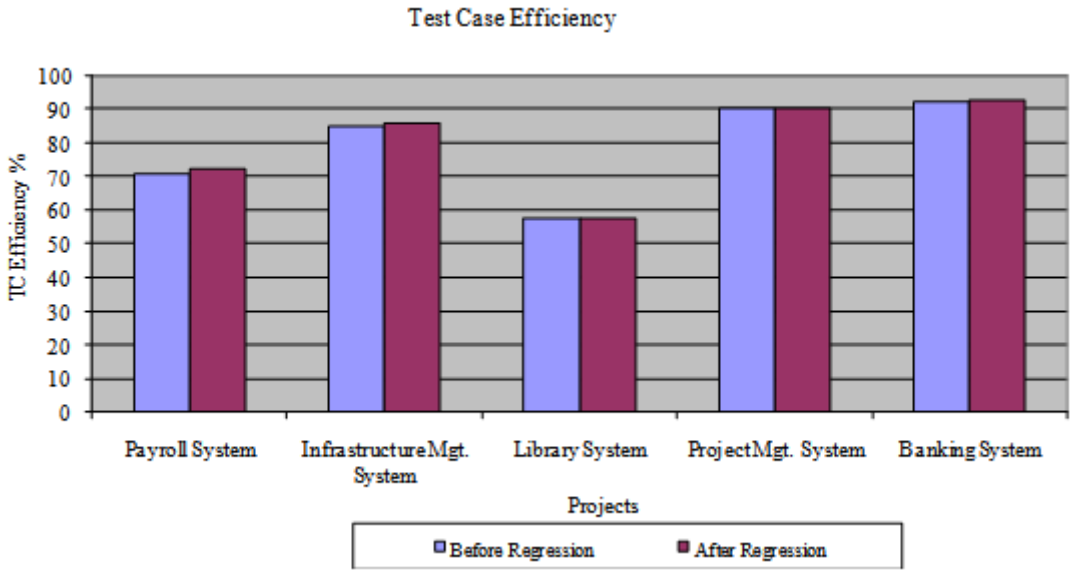


Fig 10(b) Test Case efficiency for Large Projects

I.

VI. RESULT ANALYSIS AND CONCLUSION

In this research work, we have obtained the following results. When we compare project size, there is an increase in code size. Also when we compare the test suite volume size, it is shown that number of test cases also increased for testing the modified code. These factors also have shown that the testing metrics such as defect density, test case efficiency also increase for many of the programs that we considered. Also it is shown that the existing test suite may not be good enough to test the modified code thereby we need to add new set of test cases.

REFERENCES

[1] Kan, S. H. Metrics and Models in Software Quality Engineering, Second Edition, Addison-Wesley, Boston, 2004.

[2] Konda, K. R. "Measuring Defect Removal Accurately, Software Test & Performance," Vol. 2, No. 6, July, 2005, pp. 35-39.

[3] Gregg Rothermel, Sebastian Elbaum, Alexey G. Malishevsky, Praveen Kallakuri and Xuemei Qiu, "On test suite composition and cost-effective regression testing", ACM Transactions on Software Engineering and Methodology (TOSEM) TOSEM Homepage archive, Volume 13 Issue 3, Pp. 277 – 331, July 2004.

[4] Mrinal Kanti Debbarma, Nagendra Pratap Singh, Amit Kr. Shrivastava and Rishi Mishra, " Analysis of Software Complexity Measures for Regression Testing", ACEEE Int. J. on Information Technology, Vol. 01, No. 02, Sep 2011.

[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] R. Kavitha and N. Sureshkumar, "Test Case Prioritization for Regression Testing based on Severity of Fault," International Journal on Computer Science and Engineering, Vol. 2, No. 5, pp.1462-1466, 2010.

[7] Roya Alavi and Shahriar Lotfi, " The New Approach for Software Testing Using a Genetic Algorithm Based on Clustering Initial Test Instances", proc. of International Conference on Computer and Software Modeling IPCSIT vol.14 (2011) © (2011) IACSIT Press.

[8] Pakinam N. Boghdady, Nagwa L. Badr, Mohamed Hashem and Mohamed F.Tolba, " Test Case Generation and Test Data Extraction Techniques", International Journal of Electrical & Computer Sciences IJECS-IJENS Vol: 11 No: 03, 113803-9191 © June 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.