International Journal of Computational Intelligence Research
ISSN 0973-1873 Volume 13, Number 2 (2017), pp. 197-211

http://www.ripublication.com

Phase wise Cost Effective Optimal Prioritization for System Regression Testing

Ms. Aman Hooda[1] and Dr. Ankit Kumar[2]

[1,2] Department of Computer Science
B.M University, Rohtak, Haryana, India.

Abstract

Continual enhancements are a prerequisite for any software application; because of which every organization confines Continuous Improvement Process (CIP). Statistical methods are used to calculate effort and cost required for regression testing. It is a perpetual attempt on behalf on an organization to refine its development processes, ennoble its products and outfit their services to meet ever-changing and elevating business objectives. This research has envisioned a distinct Design Phase-established architecture for the regression test case prioritization enigma and appraised its realization. The proposed technique is further bifurcated into two main phases Preference and Reordering. We attained experimentally that how design based prioritization requiring sufficient testing resources and covering whole coverage along with recognizing most risky bugs at the most primitive stage can be carried successfully in resource limitation environment.

Keywords*:* Preference, Reordering, Continuous Improvement Process and Statistical methods.

I. INTRODUCTION

Software systems are developed to serve particular requirements of both a business and a technical nature. In today's competitive business environment, developing a quality software product along with effective length of time it takes from a product being conceived until its being available in market is a determining factor for project success. The ease of extensibility, compliance and applicability of the software systems is the core reason why a large number of software is being interspersed in large convoluted systems as well as accustomed systems [2]. We realize that modulations are cardinal to software; each software experiences modification during

its life time; before deployment as well as after deployment [11]. Usually a change is made to "enhance" the application by adding new attributes and services, during bug fixing or adaptation to newer execution environments. In general most software application requires sustentation due to depreciation caused by use. Faults make software systems to fail, causes user and stakeholders aggravation and cost business organization business money. We need an optimal amount of testing based on the risk assessment of the application [1].

It is evident from the figure below the actual expenses incurred usually surmount the planned cost values, inundating the preordained earned values of budget.

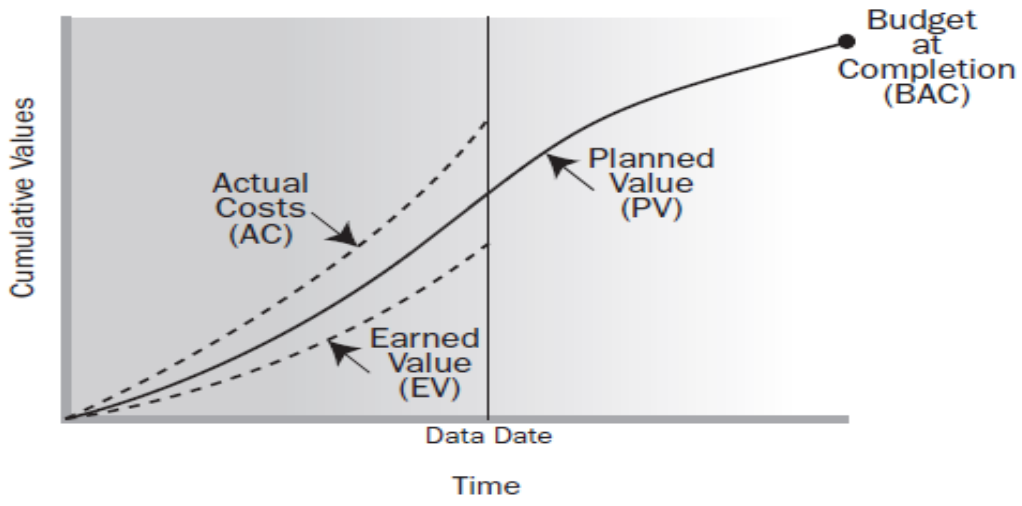

Figure 1 Relationships between AC, EV, PV and BAC; source: PMBOK, 4th Ed.[15]

The rest of the paper is organized as follows: Section II reviews the various regression testing techniques and related work. Section III describes the proposed approach to cost effective regression testing with constrained resources. Section IV describes the Empirical studies and results of the proposed approach. Section V concludes and discusses future work.

II. REVIEW OF LITERATURE

The technique of prioritization was introduced by Wong et al. in a hybrid technique [16]. The proposed prioritization technique relied upon the fact that greater the outline attained by the test cases, more is the probability of exposing the defects earlier during the regression testing process. . It has been tested empirically by Jeffrey and Gupta [18] to optimize the time and cost spent on testing, which makes the test case prioritization important. Regression TCP approaches intended by Kim and Porter [3] used structural measures to choose the test cases. In many applications, test cases prioritization approach intended by Sebastian Elbaum [17] was done on the basis of costs and fault metrics. To make the prioritization cost fruitful, it is very important to choose the appropriate prioritization approach based on the testing situations. Gordon

and Franz [6] substantiated test-case prioritization with the use of model-checkers. They presented that test case prioritization can be performed automatically during test-case generation, without post-processing, thus removing the need for test-suite post-processing. Jussi Kasurinen et al.[9] indicated that the basic approaches to test case selection are usually oriented towards two possible objectives. One is the risk-based and other is design-based selection and prioritization. Ryan Caulson et al.[10] implemented a new prioritization technique that incorporated a clustering approach and utilized code coverage, code complexity and history data on real faults that have been reported by users after the product has been released.

III. PROPOSED APPROACH

For experimentation and analysis, we considered new loan account from banking software system. The below diagram depicts the wizard screens used in creation of a loan account:

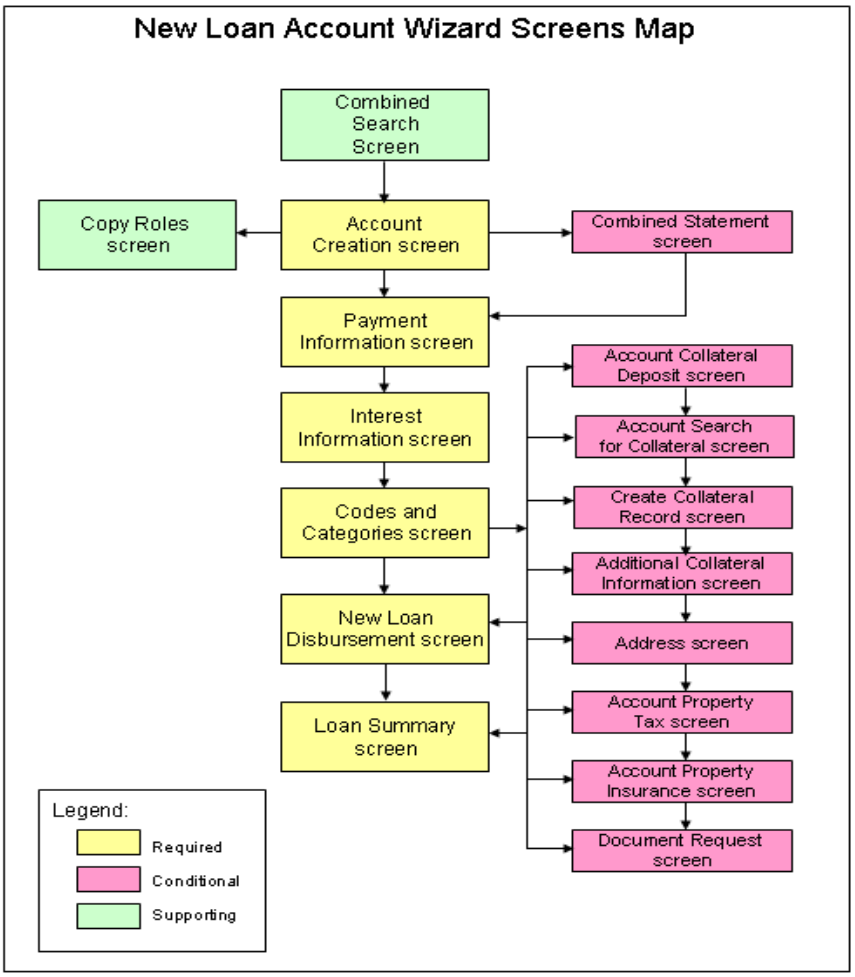

Fig 2 New Loan Wizards

Regression testing is absolutely imperious to ensure the quality of the software system after revisions. The only detriment to regression testing is substantial expansion of test suite with each subsequent version. The necessity to improve the efficiency of regression testing is highly desired. This can be achieved by using relevant regression testing strategies and techniques while adhering to endeavor the predetermined objective against which the success of regression testing can be measured. Test case prioritization is one expedient to reduce the cost. In proposed approach we are attempting to generate a superlative sequel of regression test suite using clustering approach based on design phase and comparing it with cost incurred while carrying out verification during system testing.

Our proposed approach is depicted as-

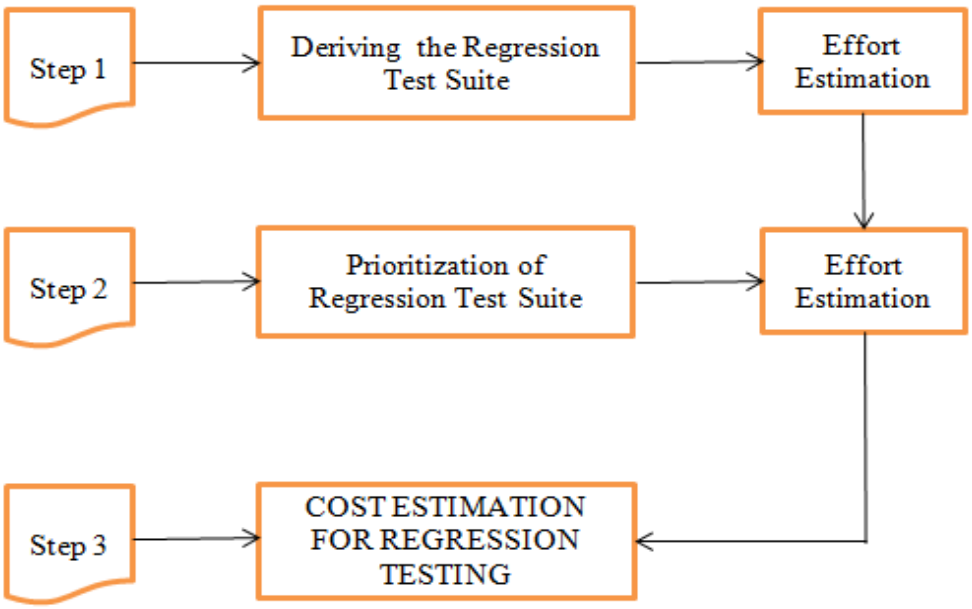

Figure 3 Overview of Proposed approach

Each step is further divided into sub steps which are elaborated as-

Step 1: Deriving the Regression test suite

Test Coverage has paramount significance in software testing as well as regression testing. Test coverage denotes the estimation of the competence of testing done with data with regard to different components, features and factors. It is the sum total of verification executed by a group of test cases.

For the intended research work, the total coverage is divided on the basis of Specification document; from design phase and Functionality as implied for system

testing. Each coverage domain is enumerated into five coverage areas or counts with each count addressing a related but separate detail. Different coverage areas used for this research at various phases are described as-

- Code Coverage is an estimation of extent to which the actual source code of a program is executed when a specific designated test suite is executed. It determines how much of code is being tested.
- Customer Requirement based coverage assess the measure of functional and other non functional attributes achieved when specific designated test suite is executed.
- Requirement based coverage assess the measure of functional and other non functional attributes along with conditions imperative for implementation achieved when specific designated test suite is executed.
- Condition coverage determines the degree to which each of the Boolean expression implying TRUE or FALSE have been appraised.
- Iteration coverage determines degree of measure of every repetition of each loop executed at least once.
- Fault coverage represents degree to which faults (functional & logical faults, business logic, security, GUI etc.) is identified in a condition when the expected result does not match with the actual result.
- Modification coverage represents degree to which the changes done in the software are verified.

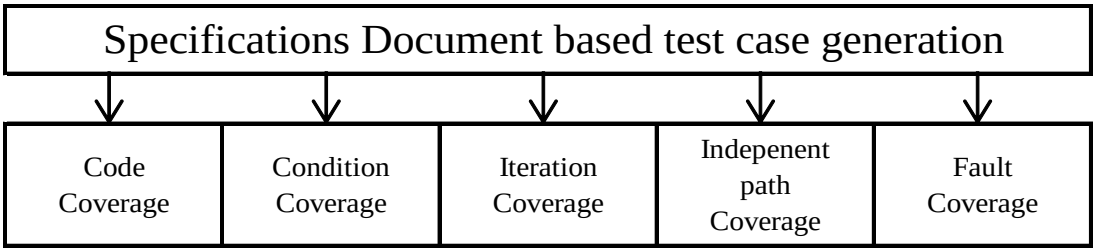

Figure 4 Deriving Design-based Regression test suite

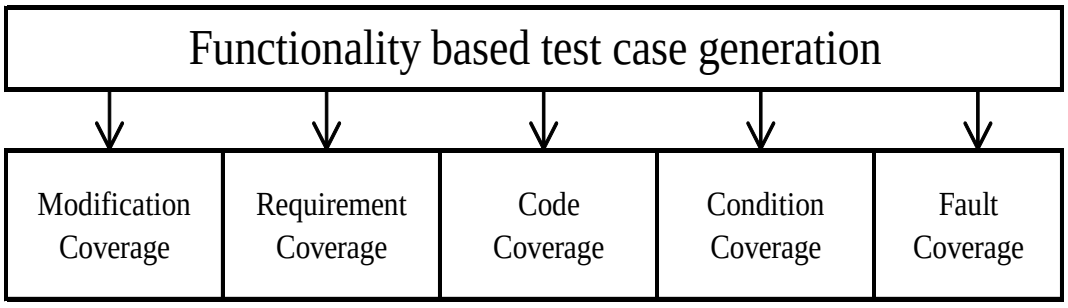

Figure 5 Functionality based test case generation

Step 2: Prioritization of Regression Test Suite

Easily, we depict the objective as one of enhancing our test suite's fault detection rate. There are various intensions for meeting this objective. An enhanced fault detection rate during regression testing can let software engineers initiate their debugging activities prior than might otherwise be promising, pacing the delivery of the software

An enhanced fault detection rate can also give faster response on the system under test, and provide prior confirmation when quality intensions have not been met, granting planned decisions about delivered schedules to be made prior than might otherwise be promising. Further, in a testing condition in which the sum of testing time that will be vacant is uncertain such prioritization can improve the likelihood that whenever the testing process is halted, testing resources will have been used up more cost-effectively in relation to possible fault detection than they might otherwise have been.

Test case prioritization approaches list test cases for implementation in a sequence that endeavors to increase their usefulness in meeting some implementation goal. Different goals are possible; one includes fault detection rate that is a standard of how quickly faults are noticed within the testing procedure. An enhanced fault detection rate during testing can bestow faster response on the system under test, and let software engineers start correcting faults prior than might otherwise be feasible.

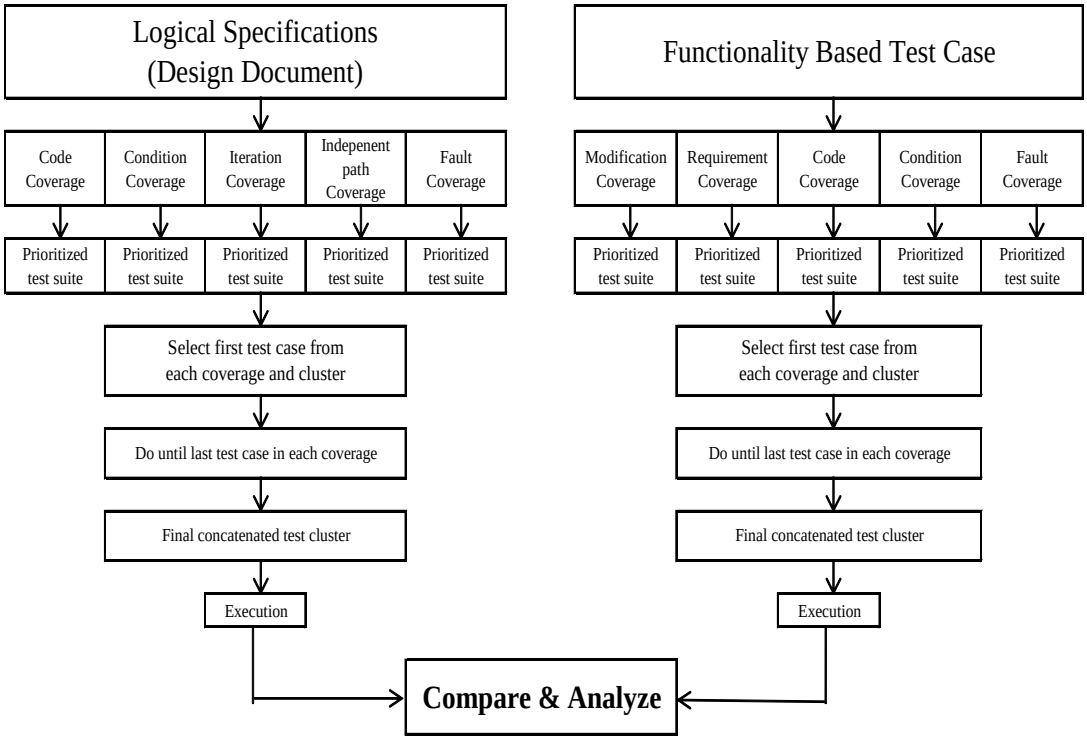

Figure 7 Overview of Proposed Research Methodology

3.1 Calculating Weights for Test Case Prioritization

For calculating the weights for individual test case following factors are considered:-

3.1.1 Fault Detection Ratio (FDR): Fault Detection Ratio (FDR) is defined as the average number of defects found per minute by a test case. The rate of fault detection of test case x has been calculated using the number of faults detected and the time taken to find out those faults for each test case of test suite.

$$FDRx = \frac{Nx}{tx} \tag{1}$$

Where

Nx represents total no. of bugs found by a test case Tx

tx represents total time taken by test case Tx to find Nx bugs.

3.1.2 Risk Detection Ratio (RDR): It can be defined as the ability of test case to detect severe faults per unit time. Testing effectiveness can be improved by focusing on the test case that is likely to detect high number of severe faults at the earliest. So, for each fault severity value was assigned based on impact of the fault on the system. Severity value has been assigned based on a 10 point scale. Total severity value for each test case may be calculated as

$$TSx = \sum_{n=1}^{10} \text{SVx} \tag{2}$$

Hence Risk Detection Ratio can be calculated as –

$$RDRx = \frac{SVx * \text{Nx}}{tx} \times 10 \tag{3}$$

3.1.3 Test Case Weight Value: Test case weight value of i^{th} test case is computed as followed-

$$\text{Priority Weight} = FDRx + RDRx \tag{4}$$

Test cases are sorted for execution based on the descending order of Priority Weight (PW), such that test case with highest PW value is placed at top in the test suite they belong.

Step 3: Cost Estimation for Regression Testing

Overall cost of regression testing can be calculated as-

a) Calculate Total number of Coverage areas (TCA)
b) Calculate total number of test cases (TTC). It can be calculated as Total no. of Coverage area × Avg. test cases per Coverage area.

c) Calculate Total Execution Time (TET) for a test suite of Coverage domain
d) Calculate Test Case Generation Time (TCGT). It is usually 1.5 times of TET.
e) Calculate Test Case Processing Time (TCPT). Its value is either 0 to 5 times of TET
f) Calculate Report Generation Time (RGT). It is usually 0.2 times of TET.
g) Calculate Test Environment Setup Time (TEST)

Total Regression Cost (TRC) = TET + TCGT + TCPT + RGT + TEST + Buffers (5)

3.2 Proposed Validation Metric

Two new metrics one based on fault detection rate and another based on test case costs are devised to validate the proposed partition cluster based test case prioritization technique.

$$\text{Fault Excision Efficiency (EFF)} = \frac{Ddp}{Ddp+Dfn} \times 100 \quad (6)$$

Where
Ddp represents total no. of faults detected by test cases generated in Design phase.
Dfn represents total no. of faults detected by test cases generated for functionality.

Fault Diligence Estimation (FDE) = Total No. of faults resolved/ Total effort spent (7)

IV. EXPERIMENTATION

For experimentation and analysis, we generated and processed test cases according logical specifications while updating design. An experiment was conducted to perform regression testing by inserting faults with different severities in the module, and finally time required by every test case to detect faults have noted for each one of them. Table 1 represents the priority weight values for tests case based on logical specifications and Table 2 represents the award values for tests case based on functionality (as achieved after implementation of software system).

The values of rate of fault detection ratio (FDR), and risk detection ratio (RDR) for test cases is calculated by using equation (1), equation (3) and Priority Weight (PW) is calculated using equation (4) respectively. Table 3, 4 represents the cluster formed by selecting test case from each group of prioritizing criteria of the logical specification and functionality respectively.

Table 1 Logical Specification (Design Phase) based Priority weights

Test cases & associated faults for Code Coverage							
Test case	No of Faults detected	Time taken	Severity	Total Severity	Fault Detection Ratio (FDR)	Risk Detection Ratio (RDR)	Priority Weight (FDR+RDR)
T4	4	41	4+6+6+6	22	0.097561	21.463415	21.560976
T2	3	29	6+6+8	20	0.1034483	20.689655	20.793103
T7	3	36	6+6+8	20	0.0833333	16.666667	16.75
T6	4	55	4+4+6+6	20	0.0727273	14.545455	14.618182
T3	2	28	6+4	10	0.0714286	7.1428571	7.2142857
T5	2	24	6+2	8	0.0833333	6.6666667	6.75
T1	3	49	2+4+4	10	0.0612245	6.122449	6.1836735
T8	1	14	6	6	0.0714286	4.2857143	4.3571429
T9	2	15	6+6	2	0.1333333	2.6666667	2.8
Test cases & associated faults for Independent Path Coverage							
Test case	No of Faults detected	Time taken	Severity	Total Severity	Fault Detection Ratio (FDR)	Risk Detection Ratio (RDR)	Priority Weight (FDR+RDR)
T13	3	32	4+2+8	14	0.09375	13.125	13.21875
T12	4	56	2+4+4+8	18	0.0714286	12.857143	12.928571
T18	2	18	4+6	10	0.1111111	11.111111	11.222222
T15	2	23	6+6	12	0.0869565	10.434783	10.521739
T11	3	36	4+2+6	12	0.0833333	10	10.083333
T14	2	24	4+6	10	0.0833333	8.3333333	8.4166667
T16	2	29	4+8	12	0.0689655	8.2758621	8.3448276
T17	3	47	4+2+6	12	0.0638298	7.6595745	7.7234043
T10	2	33	4+6	10	0.0606061	6.0606061	6.1212121
Test cases & associated faults for Fault Coverage							
Test case	No of Faults detected	Time taken	Severity	Total Severity	Fault Detection Ratio (FDR)	Risk Detection Ratio (RDR)	Priority Weight (FDR+RDR)
T19	3	34	4+4+6	14	0.0882353	12.352941	12.441176
T27	2	20	4+8	12	0.1	12	12.1
T21	2	26	4+6	10	0.0769231	7.6923077	7.7692308
T26	2	37	6+8	14	0.0540541	7.5675676	7.6216216
T24	2	24	6+2	8	0.0833333	6.6666667	6.75
T22	3	46	4+4+2	10	0.0652174	6.5217391	6.5869565
T25	3	49	2+4+4	10	0.0612245	6.122449	6.1836735
T20	2	28	4+4	8	0.0714286	5.7142857	5.7857143
T23	2	39	2+4	6	0.0512821	3.0769231	3.1282051
Test cases & associated faults for Condition Coverage							
Test case	No of Faults detected	Time taken	Severity	Total Severity	Fault Detection Ratio (FDR)	Risk Detection Ratio (RDR)	Priority Weight (FDR+RDR)
T30	3	18	4+6+2	12	0.1666667	20	20.166667
T28	3	33	4+4+6	14	0.0909091	12.727273	12.818182
T31	2	15	4+2	6	0.1333333	8	8.1333333
T34	2	25	4+6	10	0.08	8	8.08
T36	2	26	4+6	10	0.0769231	7.6923077	7.7692308
T32	3	90	8+8+6	22	0.0333333	7.3333333	7.3666667
T33	1	9	6	6	0.1111111	6.6666667	6.7777778
T29	2	24	4+4	8	0.0833333	6.6666667	6.75
T35	2	28	4+4	8	0.0714286	5.7142857	5.7857143

Test cases & associated faults for Iteration Coverage							
Test case	No of Faults detected	Time taken	Severity	Total Severity	Fault Detection Ratio (FDR)	Risk Detection Ratio (RDR)	Priority Weight (FDR+RDR)
T45	2	16	4+4	8	0.125	10	10.125
T44	2	31	8+6	14	0.0645161	9.0322581	9.0967742
T41	2	38	6+8	14	0.0526316	7.3684211	7.4210526
T43	2	28	6+4	10	0.0714286	7.1428571	7.2142857
T40	2	34	8+4	12	0.0588235	7.0588235	7.1176471
T39	2	27	4+4	8	0.0740741	5.9259259	6
T38	2	29	6+2	8	0.0689655	5.5172414	5.5862069
T42	1	15	6	6	0.0666667	4	4.0666667
T37	1	18	6	6	0.0555556	3.3333333	3.3888889

Table 2 Functionality based Priority weights

Test Cases and associated faults for Modification Coverage							
Test case	No of Faults detected	Time taken	Severity	Total Severity	Fault Detection Ratio (FDR)	Risk Detection Ratio (RDR)	Priority Weight (FDR+RDR)
T1	2	15	2+8	10	0.1333333	13.333333	13.466667
T7	2	19	4+4	8	0.1052632	8.4210526	8.5263158
T6	2	36	4+2	6	0.0555556	4.8	4.8555556
T3	1	10	4	4	0.1	4	4.1
T4	2	23	2+2	4	0.0869565	3.4782609	3.5652174
T9	2	24	2+2	4	0.0833333	3.3333333	3.4166667
T2	2	36	2+4	6	0.0555556	3.3333333	3.3888889
T5	2	29	2+2	4	0.0689655	2.7586207	2.8275862
T8	2	34	2+2	4	0.0588235	2.3529412	2.4117647
Test Cases and associated faults for Requirement Coverage							
Test case	No of Faults detected	Time taken	Severity	Total Severity	Fault Detection Ratio (FDR)	Risk Detection Ratio (RDR)	Priority Weight (FDR+RDR)
T11	2	20	2+4	6	0.1	6	6.1
T10	2	14	2+2	4	0.1428571	5.7142857	5.8571429
T14	2	16	2+2	4	0.125	5	5.125
T16	2	24	4+4	8	0.0833333	4.7058824	4.7892157
T15	1	14	6	6	0.0714286	4.2857143	4.3571429
T17	1	7	2	2	0.1428571	2.8571429	3
T18	2	32	2+2	4	0.0625	2.5	2.5625
T13	1	10	2	2	0.1	2	2.1
T12	1	14	2	2	0.0714286	1.4285714	1.5
Test Cases and associated faults for Code Coverage							
Test case	No of Faults detected	Time taken	Severity	Total Severity	Fault Detection Ratio (FDR)	Risk Detection Ratio (RDR)	Priority Weight (FDR+RDR)
T24	2	21	4+4	8	0.0952381	7.6190476	7.7142857
T21	2	18	2+4	6	0.1111111	6.6666667	6.7777778
T19	2	26	6+2	8	0.0769231	6.1538462	6.2307692
T25	2	25	4+2	6	0.08	4.8	4.88
T22	2	26	4+2	6	0.0769231	4.6153846	4.6923077
T27	2	18	2+2	4	0.1111111	4.4444444	4.5555556
T26	2	22	2+2	4	0.0909091	3.6363636	3.7272727
T23	1	17	4	4	0.0588235	2.6666667	2.7254902
T20	1	15	2	2	0.0666667	1.3333333	1.4

Test Cases and associated faults for Condition Coverage

Test case	No of Faults detected	Time taken	Severity	Total Severity	Fault Detection Ratio (FDR)	Risk Detection Ratio (RDR)	Priority Weight (FDR+RDR)
T32	2	23	2+6	8	0.0869565	6.9565217	7.0434783
T34	3	30	2+2+2	6	0.1	6	6.1
T28	1	11	6	6	0.0909091	5.4545455	5.5454545
T36	2	27	2+4	6	0.0740741	4.137931	4.2120051
T33	2	30	2+4	6	0.0666667	4	4.0666667
T30	2	31	2+4	6	0.0645161	3.8709677	3.9354839
T35	2	27	2+2	4	0.0740741	2.962963	3.037037
T29	2	29	2+2	4	0.0689655	2.7586207	2.8275862
T31	1	12	2	2	0.0833333	1.6666667	1.75

Test Cases and associated faults for Fault Coverage

Test case	No of Faults detected	Time taken	Severity	Total Severity	Fault Detection Ratio (FDR)	Risk Detection Ratio (RDR)	Priority Weight (FDR+RDR)
T42	3	40	2+6+4	12	0.075	9	9.075
T40	2	29	6+2	8	0.0689655	5.5172414	5.5862069
T44	2	33	4+4	8	0.0606061	4.8484848	4.9090909
T37	2	28	2+4	6	0.0714286	4.2857143	4.3571429
T41	2	31	2+4	6	0.0645161	3.8709677	3.9354839
T38	1	16	6	6	0.0625	3.75	3.8125
T45	2	27	2+2	4	0.0740741	2.962963	3.037037
T43	2	34	2+2	4	0.0588235	2.3529412	2.4117647
T39	1	19	4	4	0.0526316	2.1052632	2.1578947

Table 3. Clustering achieved for Logical Specification

	Code coverage	Independent path coverage	Fault coverage	Condition coverage	Iteration coverage	PW per cluster
Cluster 1	21.56097561	13.21875	12.441176	20.16666667	10.125	77.512569
Cluster 2	20.79310345	12.92857143	12.1	12.81818182	9.096774194	67.736631
Cluster 3	16.75	11.22222222	7.7692308	8.133333333	7.421052632	51.295839
Cluster 4	14.61818182	10.52173913	7.6216216	8.08	7.214285714	48.055828
Cluster 5	7.214285714	10.08333333	6.75	7.769230769	7.117647059	38.934497
Cluster 6	6.75	8.416666667	6.5869565	7.366666667	6	35.12029
Cluster 7	6.183673469	8.344827586	6.1836735	6.777777778	5.586206897	33.076159
Cluster 8	4.357142857	7.723404255	5.7857143	6.75	4.066666667	28.682928
Cluster 9	2.8	6.121212121	3.1282051	5.785714286	3.388888889	21.22402

Table 4 Clustering achieved for Functionality

	Modification Coverage	Requirement Coverage	Code Coverage	Condition Coverage	Fault Coverage	PW per cluster
Cluster 1	13.46666667	6.1	7.7142857	7.043478261	9.075	43.399431
Cluster 2	8.526315789	5.857142857	6.7777778	6.1	5.586206897	32.847443
Cluster 3	4.855555556	5.125	6.2307692	5.545454545	4.909090909	26.66587
Cluster 4	4.1	4.789215686	4.88	4.212005109	4.357142857	22.338364
Cluster 5	3.565217391	4.357142857	4.6923077	4.066666667	3.935483871	20.616818
Cluster 6	3.416666667	3	4.5555556	3.935483871	3.8125	18.720206
Cluster 7	3.388888889	2.5625	3.7272727	3.037037037	3.037037037	15.752736
Cluster 8	2.827586207	2.1	2.7254902	2.827586207	2.411764706	12.892427
Cluster 9	2.411764706	1.5	1.4	1.75	2.157894737	9.2196594

V. RESULT & ANALYSIS

The Bar chart below clearly presents that the Fault Detection Ratio(FDR) of test cases generated and prioritized while updating Design Phase is substantially high as compared to test cases generated for System testing after all modification and updation achieved.

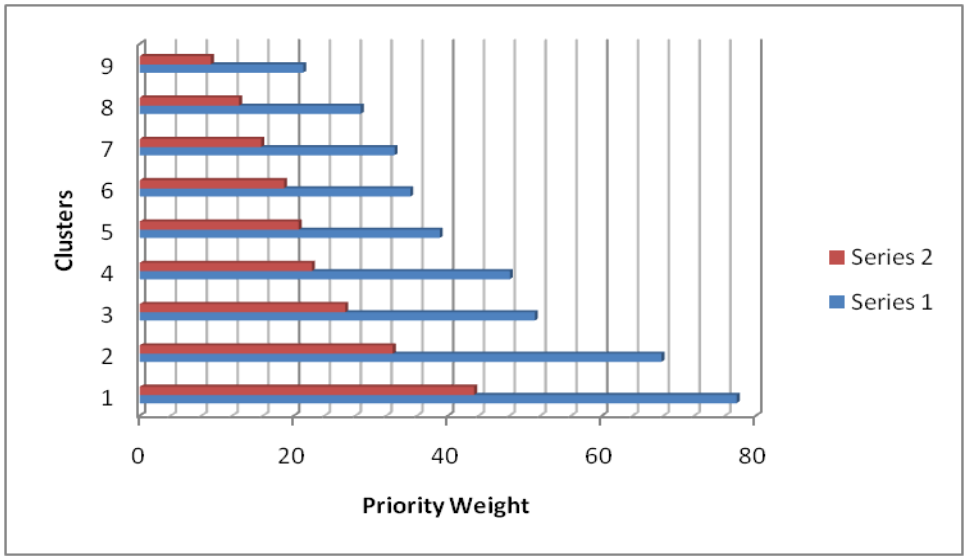

Figure 8 FDR comparison

In the figure above, series 1 represents Fault Detection Ratio (FDR) of test cases generated and prioritized while updating Design Phase and series 2 represents Fault Detection Ratio (FDR) of test cases generated and prioritized while updating System testing after all modification and updation achieved.

Table 5 Total costs Estimations for regression testing

Cost Factors	Costs Incurred for Design Phase based Test Case Generation & Prioritization	Costs Incurred for Functionality based Test Case Generation & Prioritization
a) Total number of Coverage areas (TCA)	5	5
b) Total number of test cases (TTC). It can be calculated as Total no. of Coverage area × Avg. test cases per Coverage	5*9 = 45	5*9 = 45
c) Total Execution Time (TET) for a test suite of Coverage domain	23.26 Hrs	17.36 Hrs
d) Test Case Generation Time (TCGT). It is usually 1.5 times of TET.	1.5 * 23.26 = 34.89 Hrs	1.5 * 17.36 = 26.04 Hrs
e) Test Case Processing Time (TCPT). Its value is either 0 to 1.5 times of TET	0 * 23.26 = 0 Hrs	1.2 * 17.36 = 20.83 Hrs
f) Report Generation Time (RGT). It is usually 0.2 times of TET.	0.2 * 23.26 = 4.65 Hrs	0.2 * 17.36 = 3.37 Hrs
g) Calculate Test Environment Setup Time (TEST)	25 Hrs	25 Hrs

Total cost in terms of Time taken by Test Cases generated & Prioritized at Design Phase is calculated as: TET + TCGT + TCPT + RGT + TEST.

Costs Incurred for Design Phase based Test Case Generation & Prioritization	23.26 + 34.89 + 0 + 4.65 + 25 = **87.8 Hrs**
Costs Incurred for Functionality based Test Case Generation & Prioritization	17.36 + 26.04 + 20.83 + 3.37 + 25 = **92.7 Hrs**

Calculating EFF for Design Phase based test case generation = [104/ (104+81)] * 100
(104/185) * 100 = 56.26%

Calculating EFF for Functionality based test case generation = [81/ (104+81)] * 100
(81/185) * 100 = 43.78%

Calculating FDE for Design Phase based test case generation = 104/87.8 = 1.184

Calculating FDE for Functionality based test case generation = 81/92.7 = 0.873

VI. CONCLUSION

In this research, we have observed an implementation of a regression testing technique which used test case prioritization strategy at the design phase. The resource utilized; in terms of time is optimized and so is Prioritization while implementing our proposed approach. The prioritization technique was seen to help reduce the feedback time for concluding the regression testing.
The proposed technique is further bifurcated into two main phases

- Preference
- Reordering

In the Preference phase we were concerned with deciding the appropriate stage where we have to initiate test cases generation and fundamental factors on which prioritization will be based. In the next step of Reordering, some practical weights were assigned to conclusive factors and clustering was done on the basis of partition clustering algorithm to generate an array according to which test cases were executed.

We attained experimentally that how design based prioritization requiring sufficient testing resources and covering whole coverage along with recognizing most risky bugs at the most primitive stage can be carried successfully in resource limitation environment. The objective of this research to produce minimized regression test suite maintaining fault detection facility of the test suite admissible was well achieved and proved by experiments conducted.

In the future the work can be elaborated by implementing Dynamic Programming

REFERENCES

[1] O Mohd Yusop and S Ibrahim, "Evaluating Software Maintenance Testing Approaches to Support Test Case Evolution". International Journal on New Computer Architectures and Their Applications (IJNCAA) 1(1): 74-83, 2011

[2] Bryce RC, Colbourn CJ. Prioritized interaction testing for pair-wise coverage with seeding and constraints. Journal of Information and Software Technology 2006; 48(10):960–970.

[3] J.M Kim, A. Porter, "A History-Based Test Prioritization Technique for Regression Testing in Resource Constrained Environments," ICSE, vol. 24, pp 364-373, 2002.

[4] C. Catal, "The ten best practices for test case prioritization", ICIST, CCIS 319, pp. 452-459, 2012.

[5] W. Zhang, B. Wei, H. Du, " Test case prioritization based on Genetic Algorithm and test-points coverage," Springer International publishing, ICA3PP, LNCS 8630, pp. 644-654,2014.

[6] Fraser, Gordan, and Franz Wotawa, "Test-case prioritization with model-checkers," In 25th conference on IASTED international, 2007

[7] B. Korel, G. Koutsogiannakis, L. Tahat, "Model-Based Test Prioritization Heuristic Methods and Their Evaluation", 3rd ACM Workshop on Advances in Model Based Testing, A-MOST, 2007.

[8] Mary Jean Harrold, "Reduce, Reuse, Recycle, and Recover: Techniques for Improved Regression Testing. Proceedings, ICSM, 2009 pp.

[9] Jussi Kasurinen, Ossi Taipale and Kari Smolander, "Test case selection and prioritization: Risk-Based or Design-Based?" ACM- ESEM, 2010.

[10] Ryan Carlson, Hyunsook Do, Anne Denton, "A clustering approach to improving test case prioritization: An industrial case study." , in proceedings of the 27th IEEE International Conference on Software Maintenance (ICSM), Sept. 2011, pp 382 – 391

[11] Sheikh Umar Farooq and SMK Quadri, "Identifying some problems with selection of software testing techniques", Oriental Journal of Computer Science & Technology Vol. 3(2), 266-269 (2010.)

[12] Huang, L. and Boehm, B., "How much software quality investment is enough: A value-based approach", IEEE Software, vol 23(5), 2006, pp. 88-95, 2006.

[13] Ahlam Ansari, Anam Khan, Alisha Khan and Konain Mukadam, " Optimized Regression testing using test case Prioritization," , 7th International conference on Communication, Computing and Virtualization, pp: 152-160, March 2016

[14] Shin Yoo, Mark Harman, Paolo Tonella and Angelo Susi, "Clustering test cases to achieve Effective & Sclable Prioritization incorporating expert knowledge." ACM, ISSTA, pp: 19-23, July 2009.

[15] A Guide to the project Management body of Knowledge, 4th edition, 2008 published by ANSI- Project Management Institute. Inc.,

[16] Wong W. E., Horgan J.R., London S., and Aggarwal A. "A study of effective regression testing in practice", Proceedings of the Eighth International Symposium Software Reliability Engineering, pp. 230-238, Nov. 1997.

[17] Elbaum S., Kallakuri P., Malishevsky A., Rothermel G. and Kanduri S., "Understanding the effects of changes on the cost-effectiveness of regression testing techniques", Journal of Software Testing, Verification, and Reliability, Vol.12, No.2, pp.65- 83, 2003.

[18] Jeffrey D and Gupta N., "Test-case Prioritization using relevant slices", Proceedings of the 30th annual International Computer Software and Applications (COMPSAC), Chicago, USA, pp.18-21, September 2006.