A Novel Approach for Test Case Prioritization using Business Criticality Test Value

Arup Abhinna Acharya
School of Computer Engineering
KIIT University
Bhubaneswar

Sonali Khandai
School of Computer Engineering
KIIT University
Bhubaneswar

Durga Prasad Mohapatra
Department of Computer Science Engineering
National Institute of Technology
Rourkela

ABSTRACT
Software maintenance is an important and costly activity of the software development lifecycle. Regression testing is the process of validating modifications introduced in a system during software maintenance. It is very inefficient to re-execute every test case in regression testing for small changes. This 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. Early fault detection can provide a faster feedback generating a scope for debuggers to carry out their task at an early stage. Model Based Prioritization has an edge over Code Based Prioritization techniques. The issue of dynamic changes that occur during the maintenance phase of software development can only be addressed by maintaining statistical data for system models, change models and fault models. In this paper we present a novel approach for test case prioritization by evaluating the Business Criticality Value (BCV) of the various functions (functional and non-functional) present in the software using the statistical data. Then according to the business criticality value of various functions present in the change and fault model we prioritize the test cases are prioritized.

Keywords
Software Maintenance, Regression Testing, Test case Prioritization, Business Criticality Value

1. INTRODUCTION
Software developers often save the test suites they develop for their software, so that they can reuse those suites later as the software evolves. Such test suites are reused of regression testing [1]. Regression testing is the re-execution of some subset of test that has already been conducted. So regression testing can be defined as follows: Let P be a program and P' be a modified version of P with T be a test suite developed for P, then regression testing is concerned with validating P'. Integration testing occurs in regression testing, so number of regression tests increases and it is impractical and inefficient to reexecute every test for the software when some changes occur. For this reason, researchers have considered various techniques for reducing the cost of regression testing, like regression test selection, and test suite minimization [2, 3] etc. Regression test selection technique attempt to reduce the time required to retest a modified program by selecting some subset of the exiting test suite. Test suite minimization technique reduces testing costs by permanently eliminating redundant test cases from test suites in terms of codes or functionalities exercised. However Regression test selection and test suite minimization techniques have some drawbacks. Although some empirical evidence indicates that, in certain cases, there is little or no loss in the ability of a minimized test suite to reveal faults in comparison to the unminimized one, other empirical evidence shows that the fault detection capabilities of test suites can be severely compromised by minimization [4]. Similarly, although there is safe regression test selection techniques that can ensure that the selected subset of a test suite has the same fault detection capabilities as the original test suite, the conditions under which safety can be achieved do not always hold. So for these reasons testers may want to order their test cases or reschedule the test cases [5]. A prioritization technique schedules the test cases for execution so that the test cases with higher priority executed before lower priority, according to some criterion: Rothermal et al. [6] defines the test case problem as follows, where:

Problem: Find T' belongs to PT such that (for all T") (T" belongs to PT)(T" ≠ T') [f (T') ≥ f(T")].

T: a test suite,

PT: the set of permutations of T,

f: a function from PT to the real numbers.

Here, PT represents the set of all possible prioritization of T and f is a function that, applied to any such ordering, yields an award value for that ordering.

The objective of test case prioritization is fault detection rate, which is a measure of how quickly faults are detected during the testing process. Prioritization can again be categorized as Code Based Test Case Prioritization and Model Based Test Case Prioritization [7]. Most of the test case prioritization methods are code based. Code Based Test Case Prioritization methods are based on the source code of the system. Code based test case prioritization techniques are dependent on information relating the tests of test suite to various elements of a system's code of the original system i.e. it can utilize the information about the number of statements executed or the number of blocks of code executed. Model Based Test Case Prioritization methods are based on the system models. System modeling is widely used to model state-based system. System models are used to capture some aspects of the system behavior. Several modeling language have been developed such as Extended Finite State Machine (EFSM) and Specification Description.

Test case prioritization is ordering the test cases in a test suite to improve the efficiency of regression testing. Regression testing is a process of retesting the modified system using the old test suite to have confidence that the system does not have faults. During retesting of the system developers face the issue of ordering the tests for execution, which can be addressed using a good prioritization technique. One of the objectives of test case prioritization is 'early fault detection rate', which is a measure of how quickly faults are detected during testing process. Here a metric is used which calculates the average faults found per minute and also with the help of APFD (Average Percentage of Faults Detected) metric the

effectiveness of the prioritized and non prioritized case is compared [8,9]. 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 using the following notations: Let T = the test suite under evaluation, m = the number of faults contained in the program under test P, n = the total number of test cases And T*Fi* = the position of the first test in T that exposes fault i.

$$APFD = 1 - \frac{(TF1 + TF2 + ... + TFm)}{mn} + \frac{1}{2n}$$

But calculating APFD is only possible when prior knowledge of faults is available. Various experiment were conducted in which the rate of fault detection for each test case is calculated and order of test suite is evaluated in decreasing order of the value of rate of fault detection. Then the APFD value is determined for both the prioritized and non prioritized test suite and it is found that the APFD value of prioritized test suite is higher than the non prioritized test suite.

The rest of the paper is organized as follows: Section 2 summarizes the related works. Discussions and the analysis of our proposed methodology is given in Section 3. Section 4 presents a case study: Shopping Mall Automation System. Comparison with the related work is discussed in section 5. Section 6 contains analysis and APFD measure. The paper concludes in Section 6 and Future works are highlighted in section 7.

2. RELATED WORK

Srivastava et al. [11] proposed a prioritization technique and also used a metric called APFD (Average Percentage of Faults Detected) for calculating the effectiveness of the test case prioritization methods. The disadvantage of the method proposed is that calculation of APFD is only possible when prior knowledge of faults are available. APFD calculations therefore are only used for evaluation of effectiveness of various prioritization techniques.

Rothermel et al. [6, 8, 10] presented 21 different techniques for code based test case prioritization, which are classified into three different groups i.e. comparator group, statement level group and function level group. To measure the effectiveness of these techniques, an experiment was conducted where 7 different programs were taken. Here several dimensions like granularity were taken for test case prioritization. The main disadvantages of code based test case prioritization are it is very expensive as its execution is slow because of the execution of the actual code and code based test case prioritization may not be sensitive to the correct or incorrect information provided by the testers or the developer.

Korel et al. [13, 14] present a model based test case prioritization method which can be used for any modification of the EFSM (Extended Finite State Machine) system model. Here an experimental study is done which is used to compare the early fault detection of the various test case prioritization techniques presented in this paper. There are several model based test prioritization methods are present in this paper. Such as Selective test prioritization, Heuristic #1 test prioritization, Heuristic #2 test prioritization, Heuristic #3 test prioritization and Model dependence-based test prioritization. In selective test prioritization techniques high priority is assigned to those test cases that execute modified transitions in the modified model. A low priority is assigned to those test cases that do not execute any modified transition. And Heuristics #1, #2 and #3 have been developed for modifications with multiple marked transitions. The idea of model dependence-based test prioritization is to use model dependence analysis [15] to identify different ways in which added and deleted transitions interact with the remaining parts of the model and use this information to prioritize high priority tests. Here the authors have done a experimental study and from the experimental study it indicate that model based test prioritization techniques may improve on average the effectiveness of early fault detection as compared to random prioritization techniques.

Korel et al. [7] presented a comparison between codebased and model-based test case prioritization. The results from the experimental study indicate that model-based test prioritization detects early fault as compare to code-based test prioritization. However due to the sensitiveness property the early fault detection of model-based prioritization may be deteriorate if incorrect response is given by the tester or the developer. The model-based test case prioritization is less expensive than the code-based test case prioritization because execution of the model is faster than the execution of the whole code.

Acharya et al. [16] presented a method for prioritize the test cases for testing component dependency in a Component Based Software Development (CBSD) environment using Greedy Approach. Here the author first convert the system model i.e. sequence diagram to An Object Interaction Graph (OIG) using an algorithm. Then the OIG is traversed to calculate the total number of inter component object interactions and intra component object interactions. Depending upon the number of interactions between the object an objective function is calculated and then the test cases are ordered accordingly.

Swain et al. [17] proposed an approach to generate test cases and prioritize those test cases based on a test case prioritization metric. Here the author has used UML sequence and activity diagrams for their purpose. The sequence and activity diagrams are converted into testing flow graph (TFG) from which test cases are generated. Then the TFG is converted to a model dependency graph (MDG). Next, he calculated various weights for nodes (message-method/activity) as well as edge (condition) of the MDG based on a rational criterion. Weight of the node is calculated by using the number of nodes in Forward Slice (NFS) of node of MDG and weight of the edge is calculated by multiplication of the number of incoming control dependencies (edges) of node Ni and the number of outgoing control dependencies (edges) of node Nj. After calculating the weights of the node and edge the he calculated the weight of the basic path by adding the weight of the node and edge. Then he prioritized the test case in descending order of the weight.

Kumar et al. [18] have proposed a new approach which considers the severity of faults based on requirement prioritization. They considered four different factors to assign the weights to the requirements: Business Value Measure (BVM), Project Change Volatility (PCV), Development Complexity (DC), and Fault Proneness of Requirements. To calculate the Total percentage of fault detected (TSFD), the author used severity measure(SM) of each fault. Once the fault has been detected then they assign some severity measure to each fault according to requirement weights, to which it is mapped. Total Severity of Faults Detected (TSFD) is the summation of severity measures of all faults identified for a product.

Mall et al. [19] presented a method for model based approach to prioritize regression test cases for object-oriented programs. Here the author represents all relevant object-oriented features

such as inheritance, polymorphism, association, aggregation and exception. Here the authors also included dynamic aspects such as message path sequences from UML sequence diagrams. The author also considered the dependencies among test cases for test case prioritization. Here the author named their proposed model as Extended Object-oriented System Dependence Graph (EOSDG). This model extends LH-SDG and includes exceptions and message path sequencing information. The author named their prioritized techniques as Model-based Regression Test Case Prioritization technique. This approach involves two activity diagrams one activity diagram represent the activities that are performed before the testing process and the second activity diagram represent the activities that are performed each time a software is modified. The author constructed a backward slicing of the EOSDG n then constructed a backward slicing of the EOSDG and the collect the model elements in both the slicing and then he prioritized the test cases in descending order of the coverage of the model elements.

3. PROPOSED METHODOLOGY

In this section we discuss our proposed approach to generate a prioritize test cases. Our approach consists of the following three steps.

1) Maintaining a repository for the old/existing projects.
2) Matching the project type of the new projects with the existing projects contained in the repository and identifying the affected functions. Assigning business criticality values to the affected functions from statistical data.
3) Prioritizing the test cases according to the business criticality value of the test cases in descending order.

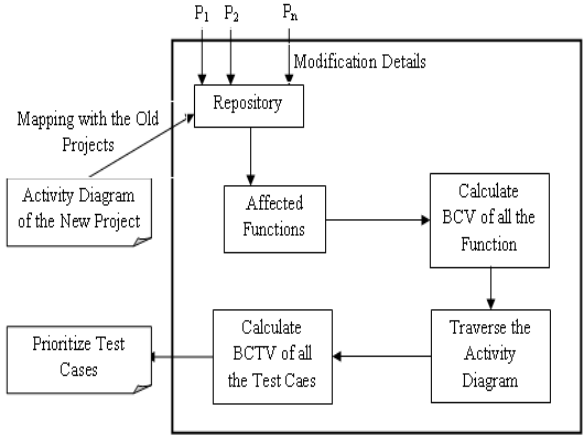

Figure 1: A Model for Model Based Test Case Prioritization

A model of the proposed methodology is shown in Fig. 1. The input to our proposed approach is an activity diagram of the new project.

As shown in Fig. 1 we first map our new project with the repository containing various projects and then we found out the affected functions due to the change in the new project are found out. After that we found out the BCV (Business Criticality Value) of each function. Then we traverse the activity graph of the new project. Using the BCV, we calculate the BCTV (Business Criticality Test Value) of each test case are found out. Finally, we prioritize the generated test cases according to BCTV of each test case.

3.1 Maintaining Repository

A repository is maintained for various numbers of projects of different category. Before maintaining the repository, a historical search is performed for finding various existing projects of unrelated categories such as application projects, networking projects, database projects, etc. for satisfying different needs of the end user. After finding out various projects belonging to different category, a table is update for each project which keeps track of the following information. This is called as repository table.

The Schema for the repository is as follows:

< Project ID, Type of Changes, Affected Function >

- First the numbers of changes that have occurred during the development of the project to satisfy the end user's requirements are stored.
- Secondly different functions i.e. both functional and non-functional that are being affected due to the changes occurred in the projects have also been maintained, which will help us in finding out the prioritized test case. The affected functions can be calculated for each change of the project with the help of foreward slicing method. We apply foreward slicing method to that particular node which are being added or modified according the end user's requirements. And the affected non-functional requirements are calculated by the expert judgment.

The affected functions have been calculated with the help of fore ward slicing algorithm which is shown in the Algorithm 1

Algorithm 1: Affected Function Calculation

Input: A activity Diagram that has a single start node and an empty set of node identifiers associated with each node.

Output: Forward slice of each node.

1. **Initialization:** Set $Si=\emptyset$ and $Vi = 0$ for $\forall i$. where Si is the set associated with node Ni and V denotes the visited status of node Ni.
2. *Call ForwardSlice(start node)*
3. ForwardSlice(node Ni)
4. begin
5. if Vi=1
6. exit(0);
7. else
8. begin Vi =1 /* Mark node as visited */
9. Find $Fi=Ni \mid N(i + 1)$ depends on Ni
10. Set $Si=Si \ U Fi$
11. for(each node $Ni \in Fi$)
12. ForwardSlice(Ni); /*Function called recursively*/
13. end
14. end if
15. end

3.2 Evaluating Business Criticality Value (Bcv)

In this section we have calculated the BCV of each function i.e. for functional and non-functional requirement. Whenever regression testing has to carry on a new project, the BCV of the various affected functions have been calculated in the following manner. There are lots of changes have been occurred such as functional changes, non-functional changes, code changes, delete some functionality, etc. Suppose a new project has encountered along with the information with us about the subsequent changes that the project has undergone according the customer requirement. Then we will first match the new project with the existing projects that are maintained in the repository. After the matching process of the new

project has been completed and a matching project has been obtained we will then find out the factors that are being affected due to the changes occurred in the new project from the repository.

In our next step we have found out the business criticality values of the various functions. A Business Criticality Value (BCV) is defining as "the extends of contribution towards the bug free application." For example in a Banking project we are having two activities such as money transaction and feedback collection, then money transaction activity has higher business criticality value than the feedback collection activity. Because the feedback collection activity is having less interaction than the money transaction activity. Business criticality value is an expertise integer value. In the similar manner the BCV is assigned to different functionalities.

The Business Critical Test Value is calculated as follows:

- First find out those factors that are being affected due to the changes made in the project.
- Find out the average interaction of each factor within that project.

The Business Critical Test Value is calculated by the following equation 1.

$$BCVfn = \frac{No.oftimesFnencounter}{Totalnooffactorbeingaffected}$$

BCV table stores certain kind of information about each factor such as the factor name along with the average interaction and the BCV value of each factor.

3.3 Prioritizing Test Cases

In this section we have prioritized our test cases according to the Business Criticality Test Values (BCTV) of the different test case. Every test case executes different factors of a project. We found the affected functionality of each test cases by using the depth first search of the project. And every factor is having different integer business critical values which have been calculated in the previous step. First we found out different test cases and then added the BCV of the encounter factor during the traversal process to find out the BCTV. By adding those values of the business criticality we find an integer value for each test case to calculate the BCTV. After that we ordered our test cases in descending order of the business criticality test values.

Prioritization Algorithm:

Our proposed technique to prioritize regression test cases is algorithmically represented in algorithm 2.

Algorithm 2

Step-1: Maintain a repository which contains different types of projects, no. of changes and the affected functionality due to the changes.

Step-2: Matching the new project with the repository and identifying the no. of changes and the affected functions respectively.

Step-3: Calculate the business criticality value of each function (functional and non-functional) according to the equation 1.

Step-4: Then traverse the activity diagram of the new coming project with the help of DFS with individual test case.

Step-5: Find the BCTV of each test case.

Step-6: Then prioritize the test cases according to the descending order of BCTV for each test case.

The prioritize table stores the information about the test cases that are obtained after the traversal process. It also stores the information about the factors that are being encountered during the traversal process. Finally we have found out the BCTV of each test case and prioritized the test suite according to the descending order of the BCV values.

4. TEST CASE PRIORITIZATION FOR A MODIFICATION IN A SHOPPING MALL AUTOMATION: A CASE STUDY

In this section a case study of our proposed approach is given. Since in our proposed method we are matching the new encounter project with the repository to find out the affected function so we have first maintain a repository..

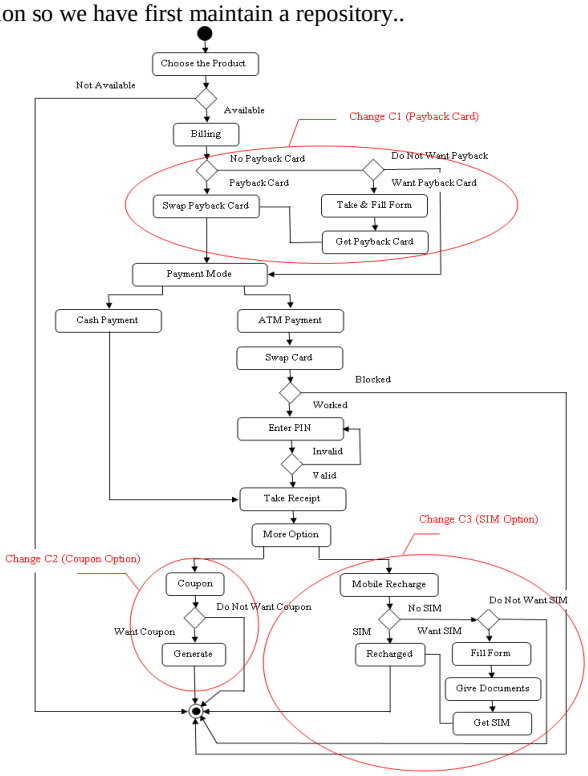

Figure 2: Activity Diagram of Big Bazaar

So that whenever a new project is encounter we can find out the affected function. After finding out the affected function from the repository the BCV value of each factor is calculated. Then the total BCV value of each test case is calculated by adding the BCV value of those functions that are visited during the DFS traversal of the Graph. Finally the test cases are prioritizing according to descending order of their BCTV value.

Suppose in the past there was a need for a big bazaar project. So a project has been designed and submitted to big bazaar. After the project have been submitted, it was found that there was a need of some additional functionality for big bazaar application so the necessary changes have been made, which activity diagram is shown in Fig 2.

A functionality detail table has been maintained which store the different types of functions with their function id as shown in Table 1. The affected functions have been calculated with the help of fore ward slicing algorithm which is shown in the Algorithm 1.

Table 1: Functionality Detail Table

FUNCTION_ID	FUCTIONALITY	FUNCTION_ID	FUCTIONALITY
F1	START	F16	Check PIN
F2	Choose the Product	F17	Take Receipt
F3	Check Available	F18	More Option
F4	Billing	F19	Mobile Recharge
F5	Have Payback Card	F20	Have SIM
F6	Want Payback Card	F21	Want SIM
F7	Swap Payback Card	F22	Fill Form
F8	Take and Fill Form	F23	Give Documents
F9	Get Payback Card	F24	Get SIM
F10	Payment Mode	F25	Recharged
F11	Cash Payment	F26	Coupon
F12	ATM Payment	F27	Want Coupon
F13	Swap Card	F28	Generate
F14	Check Working	F29	FINISH
F15	Enter PIN		

A non-functionality detail table has been maintained which store the different types of functions with their function id as shown in Table 2.

Table 2: Non-Functionality Detail Table

FUNCTION_ID	NONFUNCTIONALITY
NF3	Accessibility
NF4	Availability
NF5	Deployment
NF2	Portability
NF1	Security
NF6	Usability

The types of changes that have been made in the project with the functionalities that have been affected by the changes have been maintained in a repository which is shown in Table 3. In our project there are three numbers of changes have been made. C1 change is for payback card. C2 change is for coupon option and C3 change is for SIM card option.

Table 3: Repository (A Shopping Mall Project)

No. Of Changes	Affected Factor (Functional)	Affected Factor (Non-Functional)
C1	F5, F6, F7, F8, F9, F10, F11, F12, F13, F14, F15, F16, F17, F18, F19, F20, F21, F22, F23, F24, F25, F26, F27	NF1, NF2, NF5, NF6
C2	F18, F26, F27, F28, F29	NF1, NF3, NF4
C3	F18, F19, F20, F21, F22, F23, F24, F29	NF3, NF4, NF6

Suppose we now encounter a new shopping mall project which Activity Diagram is shown in Fig. 3.
We found that the new project matches to the big bazaar project which is store in the repository and having C1 and C2 types of changes. Now we match it with the repository and find the affected functionalities due to change C1 and C2. This is shown in Table 4.

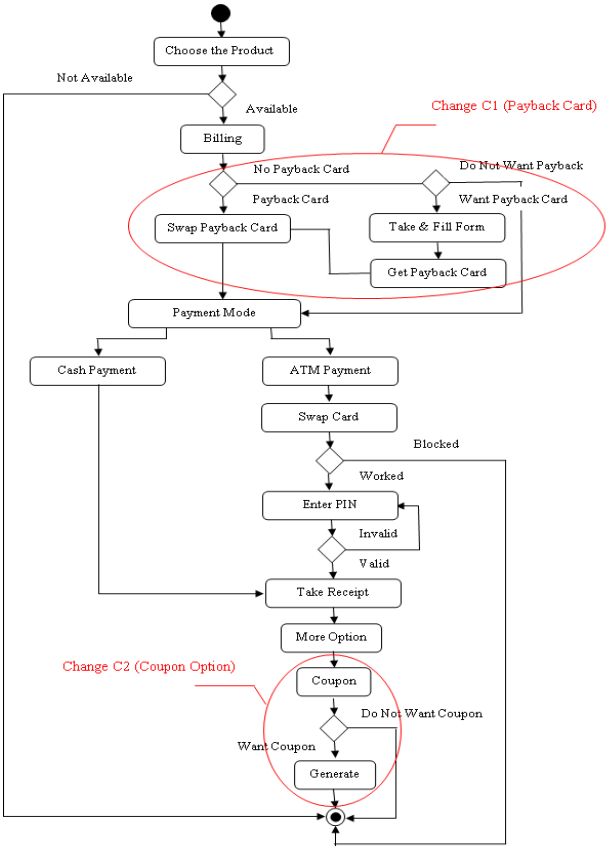

Figure 3: Activity Diagram of New Shopping Mall

Table 4: Affected Function due to the change in the new project

No. Of Changes	Affected Factor (Functional)	Affected Factor (Non-Functional)
C1	F5, F6, F7, F8, F9, F10, F11, F12, F13, F14, F15, F16, F17, F18, F19, F20, F21, F22, F23, F24, F25, F26, F27	NF1, NF2, NF5, NF6
C2	F18, F26, F27, F28, F29	NF1, NF3, NF4

Now the Business Criticality Value (BCV) of each functional requirement has been calculated according to the formula 1 and the value has been store in the Table 5.

Table 5: Business Criticality Value (BCV) Table

FUCTIO-NALITY (Fn)	No. of times each Function Encounter	BCV	FUCTIO-NALITY (Fn)	No. of times each Function Encounter	BCV
F1	0	0	F16	1	0.04
F2	0	0	F17	1	0.04
F3	0	0	F18	2	0.08
F4	0	0	F19	1	0.04
F5	1	0.04	F20	1	0.04
F6	1	0.04	F21	1	0.04
F7	1	0.04	F22	1	0.04
F8	1	0.04	F23	1	0.04
F9	1	0.04	F24	1	0.04
F10	1	0.04	F25	1	0.04
F11	1	0.04	F26	1	0.04
F12	1	0.04	F27	1	0.04
F13	1	0.04	F28	1	0.04
F14	1	0.04	F29	1	0.04
F15	1	0.04			

Now the Business Criticality Value (BCV) of each non-functional requirement has been calculated according to the formula 1 and the value has been store in the Table 6.

Table 6: Business Criticality Value (BCV) Table for Non-Functional Requirement

NonFunction ID	No. of Times Each Function Affected	BCV
NF1	2	0.33
NF2	1	0.17
NF3	1	0.17
NF4	1	0.17
NF5	1	0.17
NF6	1	0.17

Now the activity diagram shown in Fig 3 has been traversed by using to the Depth First Search (DFS) and the traversing functionalities of each test case are found out. After that the Business Criticality Test Value (BCTV) of each test cases are calculated by adding the Business Criticality Value (BCV) value of each functions and all these information have been maintained in table 7.

Table 7: Prioritization Table (Functional)

Test Case	Traversing Functionality	BCTV of each Test Case
t1	F1, F2, F3, F29	0.04
t2	F1, F2, F3, F4, F5, F6, F10, F11, F17, F18, F26, F27, F29	0.40
t3	F1, F2, F3, F4, F5, F6, F10, F11, F17, F18, F26, F27, F28, F29	0.44
t4	F1, F2, F3, F4, F5, F6, F10, F12, F13, F14, F29	0.28
t5	F1, F2, F3, F4, F5, F6, F10, F12, F13, F14, F15, F16, , F17, F18, F26, F27, F29	0.56
t6	F1, F2, F3, F4, F5, F6, F10, F12, F13, F14, F15, F16, F17, F18,F26, F27, F28, F29	0.6
t7	F1, F2, F3, F4, F5, F9, F10, F11, F17, F18, F26, F27, F29	0.40
t8	F1, F2, F3, F4, F5, F9, F10, F11, F17, F18, F26, F27, F28, F29	0.44
t9	F1, F2, F3, F4, F5, F9, F10, F12, F13, F14, F29	0.28
t10	F1, F2, F3, F4, F5, F9, F10, F12, F13, F14, F15, F16, F17, F18, F26, F27, F29	0.56
t11	F1, F2, F3, F4, F5, F9, F10, F12, F13, F14, F15, F16, F17, F18,F26, F27, F28, F29	0.6
t12	F1, F2, F3, F4, F5, F6, F7, F8, F9, F10, F11, F17, F18, F26, F27,F29	0.52
t13	F1, F2, F3, F4, F5, F6, F7, F8, F9, F10, F11, F17, F18, F26, F27,F28, F29	0.56
t14	F1, F2, F3, F4, F5, F6, F7, F8, F9, F10, F12, F13, F14, F29	0.4
t15	F1, F2, F3, F4, F5, F6, F7, F8, F9, F10, F12, F13, F14, F15, F16,F17, F18, F26, F27, F29	0.68
t16	F1, F2, F3, F4, F5, F6, F7, F8, F9, F10, F12, F13, F14, F15, F16,F17, F18, F26, F27, F28, F29	0.72

Finally the test suite is prioritized according to the descending order of the BCTV values of each test case. Hence The Prioritize Test Sequence is

t16, t15, t6, t11, t5, t10, t13, t12, t3, t8, t2, t7, t14, t4, t9, t1

The Business Criticality Test Value (BCTV) of each test case are calculated by adding the Business Criticality Value (BCV) value of each functional and non-functional requirement. And all these information have been maintained in table 8.

Table 8: Prioritization Table (Both Functional and Non-Functional)

Test Case	Traversing Functionality	BCTV of each Test Case
t1	F1, F2, F3, F29, NF1	0.37
t2	F1, F2, F3, F4, F5, F6, F10, F11, F17, F18, F26, F27, F29, NF1, NF6	0.9
t3	F1, F2, F3, F4, F5, F6, F10, F11, F17, F18, F26, F27, F28, F29, NF3, NF5	0.78
t4	F1, F2, F3, F4, F5, F6, F10, F12, F13, F14, F29, NF5	0.45
t5	F1, F2, F3, F4, F5, F6, F10, F12, F13, F14, F15, F16, F17, F18, F26, F27, F29, NF2, NF6	0.9
t6	F1, F2, F3, F4, F5, F6, F10, F12, F13, F14, F15, F16, F17, F18,F26, F27, F28, F29, NF5, NF6	0.94
t7	F1, F2, F3, F4, F5, F9, F10, F11, F17, F18, F26, F27, F29, NF1, NF2	0.9
t8	F1, F2, F3, F4, F5, F9, F10, F11, F17, F18, F26, F27, F28, F29, NF1, NF5	0.94
t9	F1, F2, F3, F4, F5, F9, F10, F12, F13, F14, F29, NF6, NF4, NF5	0.62
t10	F1, F2, F3, F4, F5, F9, F10, F12, F13, F14, F15, F16, F17, F18, F26, F27, F29, NF2, NF5	0.9
t11	F1, F2, F3, F4, F5, F9, F10, F12, F13, F14, F15, F16, F17, F18,F26, F27, F28, F29, NF1, NF2	1.1
t12	F1, F2, F3, F4, F5, F6, F7, F8, F9, F10, F11, F17, F18, F26, F27,F29, NF4, NF6	0.86
t13	F1, F2, F3, F4, F5, F6, F7, F8, F9, F10, F11, F17, F18, F26, F27,F28, F29, NF3, NF6	0.9
t14	F1, F2, F3, F4, F5, F6, F7, F8, F9, F10, F12, F13, F14, F29, NF3	0.57
t15	F1, F2, F3, F4, F5, F6, F7, F8, F9, F10, F12, F13, F14, F15, F16,F17, F18, F26, F27, F29, NF3, NF5	1.02
t16	F1, F2, F3, F4, F5, F6, F7, F8, F9, F10, F12, F13, F14, F15, F16,F17, F18, F26, F27, F28, F29, NF1, NF2, NF6	1.09

Finally the test suite is prioritized according to the descending order of the BCTV values of each test case. Hence The Prioritize Test Sequence is

t16, t11, t15, t6, t8, t2, t5, t7, t10, t13, t12, t3, t9, t14, t4, t1

5. ANALYSIS AND APFD MEASURE

In this section we have described the experimentation and analysis of our proposed methodology.

To quantify the goal of increasing a test suite's rate of fault detection, in we introduce a metric, APFD, which measures the weighted average of the percentage of faults detected over the life of the suite. APFD values range from 0 to 100; higher numbers imply faster (better) fault detection rates.

APFD can be calculated [11] using a notation:

Let T = the test suite under evaluation,

m = the number of faults contained in the program under test P,

n = the total number of test cases and

T*Fi* = the position of the first test in T that exposes fault i.

The APFD for test suite T' could be given by the Equ. 2.

$$APFD = 1 - \frac{(TF1 + TF2 + ... + TFm)}{mn} + \frac{1}{2n}$$

Table 9 shows the types of faults detected by each test case in the test suite for the example of Activity Diagram of New Shopping Mall Fig. 3.

Table 9: Types of Fault Table

Fault	Type
F1	Security Bugs
F2	Compilation Error
F3	Syntax Error
F4	Semantic Error
F5	Computing Error

Table 10 shows the number of faults detected by each test case in the test suite for the example of Activity Diagram of New Shopping Mall in Fig. 3. There are 16 number of test cases t1 through t16 in the test suit. It may be observed that for this given example, the code contains five faults, which are detected by those test cases.

Table 10: Non Prioritize Table

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T 10	T 11	T 12	T 13	T 14	T 15	T 16
F1		*				*			*		*					*
F2		*	*		*		*	*	*		*				*	*
F3	*					*	*				*		*			
F4			*					*			*				*	*
F5						*		*		*	*					*
TOTAL	1	2	2	0	1	3	2	3	2	1	5	0	1	0	2	4
TIME	6	8	12	5	7	15	15	2	8	3	12	8	12	11	4	4

Here a new test case prioritization technique is used which calculates the average faults found per minute and also with the help of APFD (Average Percentage of Faults Detected)metric the effectiveness of the prioritized and non prioritized case is compared. The effectiveness of ordering the test cases will be measured by the rate of faults detected and for this APFD metric is taken.

APFD value for Non-Prioritize Test Cases:

Now, APFD value for a non-prioritized test case (i.e. t1, t2, t3, t4, t5, t6, t7, t8, t9, t10, t11, t12, t13, t14, t15, t16) can be calculated as**,** here m = number of faults = 5 and n = number of test cases= 16. Putting the values of m, n , T*Fi* (The position of the first test case in the ordering T' of T that exposes fault i) in the Equ. 2, we get

$$APFD = 1 - \frac{(2+2+1+3+6)}{5*16} + \frac{1}{2*16}$$

APFD= 0.79

APFD value for Prioritize Test Cases(Functional Requirement):

Now let us apply Equation 2 for the prioritized test cases in case of new prioritization technique (i.e. t16, t15, t6, t11, t5, t10, t13, t12, t3, t8, t2, t7, t14, t4, t9, t1) to compute the value of APFD.

$$APFD = 1 - \frac{(1+1+3+1+1)}{5*16} + \frac{1}{2*16}$$

APFD= 0.88

APFD value for New Prioritize Test Cases:

Now let us apply Equation 2 for the prioritized test cases in case of new prioritization technique (i.e. t16, t11, t15, t6, t8, t2, t5, t7, t10, t13, t12, t3, t9, t14, t4, t1) to compute the value of APFD.

$$APFD = 1 - \frac{(1+1+2+1+1)}{5*16} + \frac{1}{2*16}$$

APFD= 0.89

Now, let us compare the APFD values for different prioritization techniques. It may be seen that the APFD value obtained for prioritized cases (using our approach) is more than non-prioritized test cases (random method). Hence, our approach generates effective prioritized test cases than the randomized approaches. Fig 4 shows the graph for both prioritize and non-prioritize test suite.

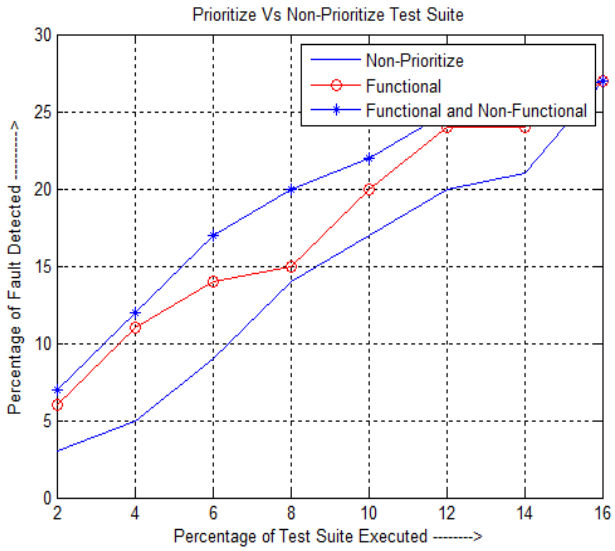

Figure 5: Comparison of Various Prioritization Techniques

6. COMPARISON WITH THE RELATED WORK

Several test case prioritization techniques are discussed in section 2. Code based test case prioritization techniques are discussed in [6, 8, 10] but as these are based on code so execution of code is very slow. Model based test case prioritization techniques are discussed in [12, 13, 14] and these are based on the no of mark transition executed by the test cases. Our approach is based on model and it considers the business criticality value of the function. As our approach is based on business criticality value so we gave more importance to that functionality whose business criticality value is more. So our approach is detecting fault as earlier to the other approach.

7. CONCLUSION

Quality of software can't be ensured without an effective testing strategy which comprises of automated test case generation, optimization of test suite design and prioritizing test cases etc. We have first gone through a literature survey

to find out the various approaches proposed by different researcher to prioritize the test cases with the help of code based as well as model based test case prioritization. We have also discussed the corresponding advantages and disadvantages of them. Our work has proposed Model- Based Test Case Prioritization for Regression Testing using Business Criticality Value for prioritizing test cases from UML activity diagrams. Majority of the test case prioritization approaches are code-based and suitable for regression testing. The proposed approach is completely model-based. In this paper we proposed a model based test case prioritization technique using the business criticality value of each functions. Business Criticality Value (BCV) is defining"as the amount of contribution towards the business of the project." The BCV of each factors both function and non-functional requirements are calculated based on the affected functionality of the project due to the subsequent changes of the project for satisfying the requirement of the customers. So the generated prioritization sequence is more efficient because it is generated based on the requirement of the customers. So the proposed prioritization method is more effective and efficient. This gives an early change to the debuggers to work with the most critical function first. Then, we compare our approach with the help of APFD method. We found that our prioritization test suite detects more faults then the non-prioritize test suite.

8. FUTURE WORK

Our approach is a model-based test case prioritization which is specifically deals with the functional and nonfunctional features of the application. The scalability of the proposed approach is yet to be tested. A suitable soft computing tool may be used to increase the effectiveness of the prioritization algorithm.

9. REFERENCES

[1] K. Onoma, W-T. Tsai, M. Poonawala, and H. Suganuma, *Regression Testing in an Industrial Environment*, Comm. ACM, vol. 41, no. 5, pp. 81-86, May 1988

[2] D. Binkley, *Semantics Guided Regression Test Cost Reduction*, IEEE Trans. Software Eng., vol. 23, no. 8, pp. 498-516, Aug. 1997.

[3] T.Y. Chen and M.F. Lau, *Dividing Strategies for the Optimization of a Test Suite*, Information Processing Letters, vol. 60, no. 3, pp. 135-141, Mar. 1996.

[4] W.E. Wong, J.R. Horgan, S. London, and A.P. Mathur, *Effect of Test Set Minimization on Fault Detection Effectiveness*, Software Practice and Experience, vol. 28, no. 4, pp. 347-369, Apr. 1998.

[5] G. Rothermel, M.J. Harrold, J. Ostrin, and C. Hong, *An Empirical Study of the Effects of Minimization on the Fault Detection Capabilities of Test Suites*, Proc. Int'l Conf. Software Maintenance, pp. 34-43, Nov. 1998.

[6] G. Rothermel, R.H. Untch, C. Chu, and M.J. Harrold, *Prioritizing Test Cases For Regression Testing*, IEEE Transactions on Software Engineering, vol. 27, No. 10, pp 929-948, October 2001.

[7] Bogdan Korel, George Koutsogiannakis, *Experimental Comparison of Code-Based and Model-Based Test Prioritization*, IEEE International Conference on Software Testing Verification and Validation Workshops, pp.77-84.

[8] G. Rothermel, R.H. Untch, C. Chu, and M.J. Harrold, *Test Case Prioritization: An Empirical Study*, Proc. Int'l Conf. Software Maintenance, pp. 179-188, Aug. 1999.

[9] Alexey G. Malishevsky, Joseph R. Ruthruff, Gregg Rothermel, Sebastian Elbaum, *Costcognizant Test Case Prioritization*, Proc. IEEE International Conference on Software Maintenance,2006.

[10] S. Elbaum, A. Malishevsky, G. Rothermel, *Test Case Prioritization: A Family of Empirical Studies*, IEEE Transactions on Software Engineering, vol. 28, No. 2, pp 159-182, 2002.

[11] Praveen Ranjan Srivastava, *Test Case Prioritization*, Journal of Theoretical and Applied Information Technology, 2005 - 2008 JATIT, pp. 178-181.

[12] H. Srikanth, L. Williams, J. Osborne, *System Test Case Prioritization of New and Regression Test Cases*, IEEE, 2005, pp.64-73.

[13] B. Korel, L. Tahat, M. Harman, *Test Prioritization Using System Models*, Proc. 21st IEEE International Conference Software Maintenance (ICSM '05), pp. 559-568, 2005.

[14] B. Korel, G. Koutsogiannakis, L. Tahat, *Application of System Models in Regression Test Suite Prioritization*, Proc. 24st IEEE International Conference Software Maintenance (ICSM '08), pp. 247-256, 2008.

[15] B. Korel, L. Tahat, B. Vaysburg, *Model Based Regression Test Reduction Using Dependence Analysis*, Proc. IEEE International Conference on Software Maintenance, pp. 214-223, 2002.

[16] A. Acharya, D. P. Mohapatra and N. Panda, *Model based Test case prioritization for testing component dependency in cbsd using uml sequence diagram*, IJACSA, vol. 1, no. 3, pp. 108-113, December. 2010.

[17] S. K. Swain, *Test Case Prioritization Based on UML Sequence and Activity Diagrams*, PhD thesis, KIIT University, 2010.

[18] Dr. V. Kumar, Sujata and M. Kumar, *Test Case Prioritization Using Fault Severity*, IJCST, vol. 1, Issue 1,, pp. 67-71, September. 2010.

[19] R. Mall and C. R. Panigrahi, *Test case prioritization of object oriented Program*, In SETLabs Brieng, Infosys, vol. 9, pp. 31-40, 2011.