Prioritizing Test Cases Using Business Criticality Test Value

Sonali Khandai
School of Computer Engineering
KIIT University
Bhubaneswar

Arup Abhinna Acharya
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 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.

I. 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) ($T" \neq T'$)

[$f(T') \geq f(T")$].

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. There are two categories of test case prioritization: general test case prioritization and version specific test case prioritization [6]. In general test case prioritization, given a program P and test suite T, we prioritize the test cases in T with the intent of finding an ordering of test cases that will be useful over a succession of subsequent modified versions of P.

Thus, general test case prioritization can be performed following the release of some version of the program during off-peak hours, and the cost of performing the prioritization is amortized over the subsequent releases. In contrast, in version-specific test case prioritization, given program P and test suite

T, we prioritize the test cases in T with the intent of finding an ordering that will be useful on a specific version P' of P. Version-specific prioritization is performed after a set of changes have been made to P and prior to regression testing P'. Because this prioritization is accomplished after P' is available. General test case prioritization does not use information about specific modified versions of P, whereas version-specific prioritization does use such information.

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 Language (SDL).

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

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

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 are given in Section 3. Section 4 presents a case study: Shopping Mall Automation System. Comparison with the related work is discussed in section 5. The paper concludes in Section 6 and Future works are highlighted in section 7.

II. 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 is 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.

Srikanth et al. [12] presented a technique that extend the code-coverage TCP techniques and apply test case prioritization at a system-level for both new and regression tests. Here the advantage is that the author uses a system level test case prioritization techniques which is called the Prioritization of Requirements for Test (PORT) based techniques.

The PORT technique prioritizes system test cases based upon four factors: requirements volatility, customer priority, implementation complexity, and fault proneness of the requirements. System level test case prioritization techniques are very beneficial because it improve the rate of fault detection of severe faults. PORT technique requires the team to conduct system analysis and write concrete test cases. The act of writing concrete test cases immediately after requirements specification can lead to the identification of ambiguous and unclear requirements, allowing requirements errors to be identified and rectified earlier.

The PORT technique allows the engineering team to monitor the requirements covered in system test; the ability to monitor requirements covered in system test is believed to be one of the challenges faced by the industry. But here only the experiment is done for four projects developed by students in advanced graduate software testing class. So the authors have to test the scalability of the PORT method.

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 i.e. random prioritization, two version of selective prioritization (version I & II) and model dependence-based test case prioritization. Here the author used RP (d), the most likely relative position of the first failed test that detects fault d to measure the early fault detection. The experimental result show that the version II of selective prioritization and model-dependence based prioritization may improve the effectiveness of the test case prioritization. But here in this paper the author used very small model for the test case prioritization, so the effectiveness of the model based test case prioritization method can't not understand properly. Also in this paper a fixed model is used i.e here the model is not changed in the system. In the further work Korel et al. [14] present model-based test case prioritization methods in which information about the system model and its behavior is used to prioritize the test suite for system retesting. 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.

III. Proposed Methodology

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

a) Maintaining a repository.

b) Matching the project type of the new projects on which regression test has to be performing with the existing projects contained in the repository and identifying the affected functions. Assigning business criticality values to the affected functions using statistical data stored in the repository.

c) Prioritizing the test cases according to the Business Criticality Test Value (BCTV) of the test cases in descending order.

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

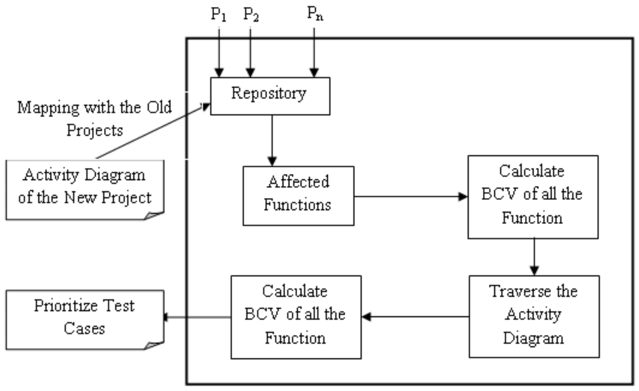
P1
P2
Pn
Mapping with the Old Projects
Repository
Activity Diagram of the New Project
Affected Functions
Calculate BCV of all the Function
Traverse the Activity Diagram
Calculate BCV of all the Function
Prioritize Test Cases

Fig. 1. Framework of the Proposed Methodology

As shown in Fig. 1 we first map our new project with the repository and then we found out the affected functions due to the change in the new project. 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.

B. Maintaining Repository

A repository is maintained for various historical projects of different category. Before maintaining the repository, a historical search is perform 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 keeping track of following information.

- First the types of changes that have occurred during the maintenance of the project satisfying the end user's requirements are identified.
- Secondly different functions 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 identified for each change of the project with the help of foreword slicing method and expertise judgments. Foreword slicing method is applied to the particular node which are being added or changed according the end user's requirements.

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

ALGORITHM1: FOREWARD SLICING ALGORITHM:

Input: An 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 $S_i = \phi$ and $V_i = 0$ for $\forall i$. where $S_i$
    is the set associated with node $N_i$ and V denotes the
    visited status of node $N_i$.
2)  Call ForwardSlice(start node)
3)  ForwardSlice (node $N_i$)
4)  Begin
5)  if $Vi=1$
6)  exit(0);
7)  else
8)  begin $Vi=1$                    /* Mark node as visited */
9)  Find $Fi = N_i \mid N_{(i+1)}$ depends on $N_i$
10) Set $S_i = S_i \cup F_i$
11) for(each node $N_i \in F_i$ )
12) ForwardSlice ($N_i$);            /*Function called recursively*/
13) End
14) end if
15) end
```

For example different projects is having C1, C2, C3...Cn types of changes which are affecting functions from the set of function. F1, F2, F3...Fn types of function. The types of changes and the list of affected functions are given in Table 1.

The Table 1 maintains three attributes such as Project ID, Types of change and affected functions. From the Table II it is found that Project-1 under gone C1, C2 and C3 types of changes. C1 is affecting F1, F2, F4, F6, F7, F8, F10, F12, F13, F15 functions, C2 is affecting F1, F5, F6, F7, F8, F9, F11, F12, F14, F15 types of functionalities and C3 is affecting F2, F3, F4, F8, F9, F10, F11, F12, F13 functionalities. In this similar manner information for other projects undergone regression test have also been maintained in the repository table.

C. Evaluating Business Criticality Value

In this section whenever regression testing has to carry on a new project, the business criticality value of the various affected functions have been calculated in the following manner. Suppose a new project has encountered along with the information with us about the subsequent changes that the project has undergone according to the customer requirement. Then we will first match the new project with the existing projects that are maintained in the repository. After finding out the matching project during the matching process we will then find out the function that are being affected due to the changes occurred in the new project from the repository.

For example we encountered a new project P NEW1 having changes C2, C7, C9 which match with the project PID_2 from the repository then we will find that function F3, F4, F5, F6, F7, F8, F10, F11, F12, F13, F15 that have been affected by the change C2, factors F2, F3, F4, F5, F8, F9, F10, F11, F12, F14, F15 have been affected by the change C7 and factors F2, F4, F6, F8, F10, F12, F14, F15 have been affected by the change C9. Since the new project P NEW1 with changes C2, C7, C9 matches with the PID 2 type. The affected functions are listed in Table 2.

Table 1. Repository Table

Project ID	Types of changes	Affected functions
PID_1	C1	F1,F2,F4,F6,F7,F8,F10,F12,F13,F15
	C2	F1,F5,F6,F7,F8,F9,F11,F12,F14,F15
	C3	F2,F3,F4,F8,F9,F10,F11,F12,F13
	C4	F1,F3,F5,F6,F7,F8,F11,F12,F13,F16
	C5	F3,F4,F6,F7,F9,F10,F12,F14,F15,F16
	C9	F6,F7,F8,F9,F10,F11,F12,F13,F14,F16
	C10	F3,F4,F5,F7,F8,F9,F13,F15
PID_2	C2	F3,F4,F5,F6,F7,F8,F10,F11,F12,F13,F15
	C3	F1,F2,F3,F6,F8,F9,F10,F12,F13,F14
	C6	F1,F2,F3,F4,F5,F6,F8,F9,10
	C7	F2,F3,F4,F5,F8,F9,F10,F11,F12,F14,F15
	C8	F1,F2,F3,F4,F5,F6,F7,F9,F10,F11
	C9	F2,F4,F6,F8,F10,F12,F14,F15
	C10	F1,F3,F5,F7,F9,F11,F13,F14,F15
	.	.
	.	.
PID_n	C1	F1,F3,F5,F7,F9,F10,F11,F12,F13,F15
	C3	F1,F4,F5,F6,F7,F9,F10,F11,F13,F14
	C5	F2,F4,F6,F7,F8,F9
	C7	F3,F4,F5,F6,F7,F8,F9,F10,F11,F14,F15
	C8	F2,F3,F4,F5,F6,F7,F8,F9,F10,F11,F15
	C9	F4,F5,F6,F7,F8,F9,F10,F11,F15
	C10	F2,F3,F4,F7,F8,F9,F10,F12,F13,F14

Table. 2. Change_Detail_New

Changes in P_NEW1	Affected Factors
C2	F3, F4, F5, F6, F7, F8, F10, F11, F12, F13, F15
C7	F2, F3, F4, F5, F8, F9, F10, F11, F12, F14, F15
C9	F2, F4, F6, F8, F10, F12, F14, F15

The Table 2 stores the information about the changes that are being made in the new project along with the functions that are being affected.

Further the Business Criticality Value (BCV) of the various functions is found out. A Business Criticality Value (BCV) is defined as the “amount of the function contribution towards the success of the project implementation.” For example in a Banking Automation Software, there are two activities such as to do transaction and collect feedback. The money transaction activity will be having higher BCV than the feedback collection activity.

The Business Critical Test Value is calculated as follows:

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

The Business Critical Value is calculated by the following formula 2.

$$\text{BCV (Fn)} = \frac{\text{No.of times Fn encounter}}{\text{Total no of factor being affected}} \qquad (2)$$

Now the BCV of each functionality have been calculated according to the formula and the value has been store in the Table 3. BCV table i.e. Table 3 stores certain kind of information such as the function name along with the average interaction and the BCV value of each function.

D. 3.3 Prioritizing Test Cases

In this section the test cases are prioritized according to the Business Criticality Values of the different factors. Every testcase executes different factors of a project according to the Depth First Search (DFS) of individual test cases are identified known as Business Criticality Test Value (BCTV) of that test case the project. And every factor is having different Business Critical Values.

Initially different test cases are identified and BCTV of the function that is encounter during the traversal process. This is shown in Table 4. Then we ordered our test cases in descending order of the Business Criticality Values.

Table.3. Business Criticality Value (BCV) Table

Factors	No. of times each factor encounter	BCV
F1	0	0
F2	2	0.067
F3	2	0.067
F4	3	0.1
F5	2	0.067
F6	2	0.067
F7	1	0.033
F8	3	0.1
F9	1	0.033
F10	3	0.1
F11	2	0.067
F12	3	0.1
F13	1	0.033
F14	2	0.067
F15	2	0.067

PRIORITIZATION ALGORITHM:

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

Algorithm 2:

1. Maintain a repository which contains different types of projects, no. of changes and the affected functionality due to the changes.
2. Matching the new project with the repository and identifying the no. of changes and the affected functions respectively.
3. Calculate the business criticality value of each function according to the equation 2.
4. Then traverse the activity diagram of the new coming project with the help of DFS with individual test case.
5. Find the BCTV of each test case by adding the BCV value of each factor.
6. Then prioritize the test cases according to the descending order of BCTV for each test case.

Table. 4. Prioritization Table

Test Case	Traversing Factors	BCTV of each Test Case
t_1	F1,F2,F3,F4,F6,F8,F9,F10,F11,F12	0.701
t_2	F1,F3,F4,F5,F7,F8,F9,F10,F15,F16,F17	0.567
t_3	F2,F3,F4,F5,F6,F7,F8,F10,F11,F12,F13,F15,F18	1.371
t_4	F3,F5,F6,F7,F8,F9,F10	0.467
t_5	F1,F3,F5,F7,F8,F9,F10,F11,F12,F14,F15	0.701

The prioritize table stores the information about the test cases that are obtained after the traversal process. It also stores the information about the functions 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 BCTV values.

Hence the prioritize test suite is:

t3, t1, t5, t2, t4 or t3, t5, t1, t2, t4.

IV. A CASE STUDY: SHOPPING MALL AUTOMATION

In this section our proposed approach is described with a case study of a Shopping Mall Automation System. Since in our proposed method we are matching the new projects goes through change with the repository to find out the affected functions. 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 function is calculated. Then the total BCTV value of each test case is calculated by adding the BCTV value of those functions that are visited during the DFS traversal of the Activity Diagram of the new encounter project. 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.

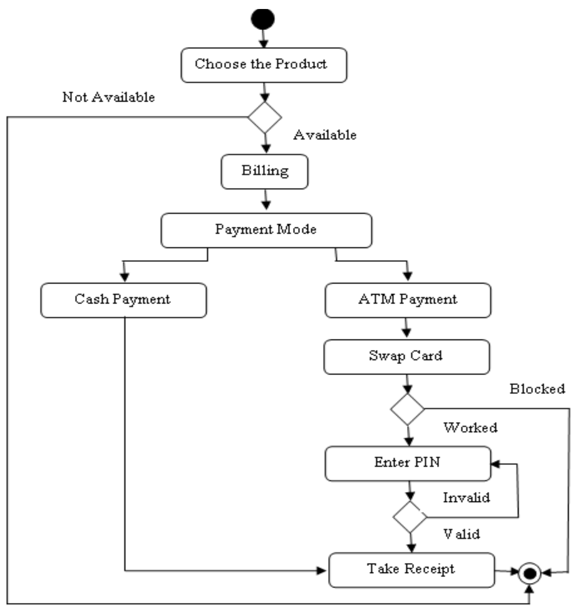

Fig. 2. Activity Diagram of Shopping Mall Automation System

So projects have been designed whose activity diagram is shown in Fig. 2.

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 to the submitted project, which activity diagram is shown in Fig. 3. A functionality detail table has been maintained which store the different types of functions present in the activity diagram of the Big Bazaar Automation System and their function id as shown in Table 5.

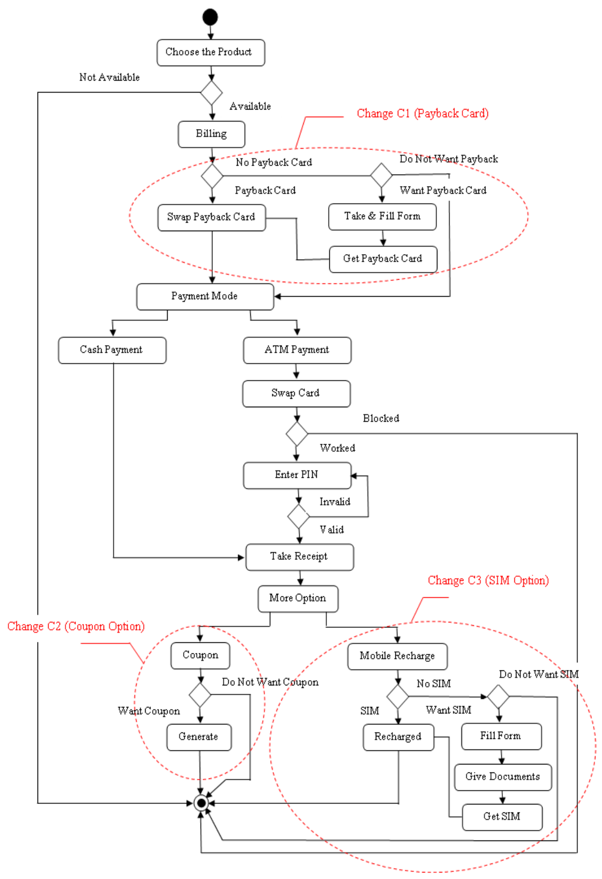

Fig. 3. Activity Diagram for A Big Bazaar Automation System

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 6.

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. The changes made are highlighted through dotted line in the activity diagram of the project which is shown in Fig. 3.

A. *Matching a project with the repository*

Suppose we now encounter a new shopping mall project which Activity Diagram is shown in Fig. 4. We found that the new project matches to the big bazaar project and having C1 and C2 types of changes those are highlighted in through dotted line in Fig. 4. Now we match it with the repository and find the affected functionalities due to the changes C1 and C2. This is shown in Table 7.

Now the BCV of each functionality have been calculated according to the formula 2 and the value has been store in the Table 8.

Table.5. 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		

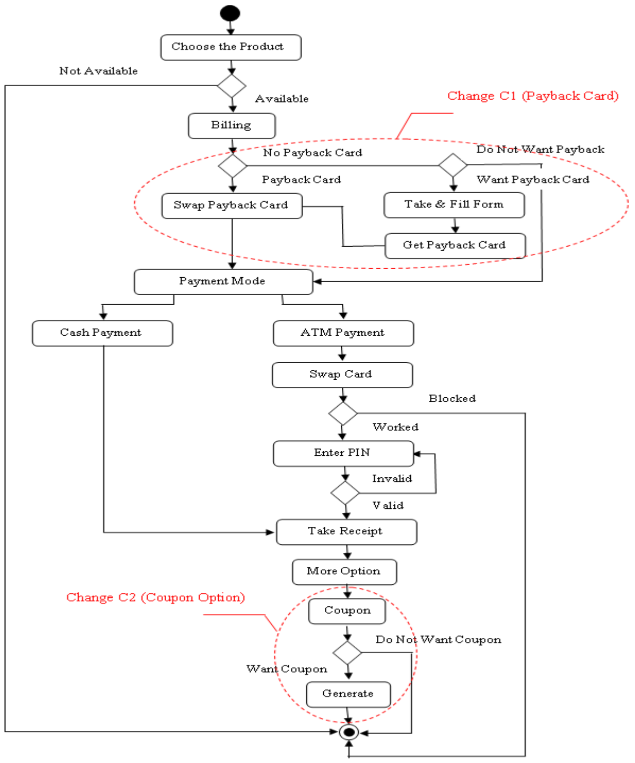

Fig. 4. Activity Diagram of New Shopping

Table. 6. Repository (A Shopping Mall Project)

No. Of Changes	Affected Functionality
C1	F5, F6, F7, F8, F9, F10, F11, F12, F13, F14, F15, F16, F17, F18, F19, F20, F21, F22, F23, F24, F25, F26, F27
C2	F18, F26, F27, F28, F29
C3	F18, F19, F20, F21, F22, F23, F24, F29

Table. 7. Affected Function due to the Change in the New Project

No. Of Changes	Affected Functionality
C1	F5, F6, F7, F8, F9, F10, F11, F12, F13, F14, F15, F16, F17, F18, F19, F20, F21, F22, F23, F24, F25, F26, F27
C2	F18, F26, F27, F28, F29

Table. 8. Business Criticality Value (BCTV) 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 activity diagram shown in Fig.4 has been traversed according to the Depth First Search (DFS) and the traversing functionalities of each test case are found out. After that the BCTV of each test cases are calculated by adding the BCV value of the functions and all these information have been maintained in table 9.

Table 9: Prioritization Table

Test Case	Traversing Functionality	BCTV of each Test Case
t_1	F1, F2, F3, F29	0.04
t_2	F1, F2, F3, F4, F5, F6, F10, F11, F17, F18, F26, F27, F29	0.40
t_3	F1, F2, F3, F4, F5, F6, F10, F11, F17, F18, F26, F27, F28, F29	0.44
t_4	F1, F2, F3, F4, F5, F6, F10, F12, F13, F14, F29	0.28
t_5	F1, F2, F3, F4, F5, F6, F10, F12, F13, F14, F15, F16, F17, F18, F26, F27, F29	0.56
t_6	F1, F2, F3, F4, F5, F6, F10, F12, F13, F14, F15, F16, F17, F18,F26, F27, F28, F29	0.6
t_7	F1, F2, F3, F4, F5, F9, F10, F11, F17, F18, F26, F27, F29	0.40
t_8	F1, F2, F3, F4, F5, F9, F10, F11, F17, F18, F26, F27, F28, F29	0.44
t_9	F1, F2, F3, F4, F5, F9, F10, F12, F13, F14, F29	0.28
t_{10}	F1, F2, F3, F4, F5, F9, F10, F12, F13, F14, F15, F16, F17, F18, F26, F27, F29	0.56
t_{11}	F1, F2, F3, F4, F5, F9, F10, F12, F13, F14, F15, F16, F17, F18, F26, F27, F28, F29	0.6
t_{12}	F1, F2, F3, F4, F5, F6, F7, F8, F9, F10, F11, F17, F18, F26, F27,F29	0.52
t_{13}	F1, F2, F3, F4, F5, F6, F7, F8, F9, F10, F11, F17, F18, F26, F27, F28, F29	0.56
t_{14}	F1, F2, F3, F4, F5, F6, F7, F8, F9, F10, F12, F13, F14, F29	0.4
t_{15}	F1, F2, F3, F4, F5, F6, F7, F8, F9, F10, F12, F13, F14, F15, F16, F17, F18, F26, F27, F29	0.68
t_{16}	F1, F2, F3, F4, F5, F6, F7, F8, F9, F10, F12, F13, F14, F15, F16, F17, F18, F26, F27, F28, F29	0.72

After calculating the BCTV value of all the test cases we than prioritized the test suite according to the descending order of the BCTV values of the test cases.

Hence The Prioritize Test Sequence is:

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

Hence, our approach generates effective prioritized test cases because it more priority to the test cases those are having more Business Criticality Value.

V. 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.

VI. Conclusion

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 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.

VII. Future Work

Our approach is a model-based test case prioritization which is specifically contains the functional features of the project. We can also extend our approach by adding the non-functional features of the project. In future we also implement our proposed approach with the help of IBM Quality Seed: Functional Tester for Regression Testing.

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.

Author Profile

Sonali Khandai is a M.tech student of KIIT University, Bhubaneswar, Odisha, INDIA. Her research areas include Software Testing. She has completed her B.tech in Computer Science and Engineering from BPUT in the year 2010. She can be reached at khandaisonali@gmail.com

Arup Abhinna Acharya is an Assistant Professor and research scholar in the School of Computer Engineering, KIIT University, Bhubaneswar, Odisha, INDIA. He received his Master's degree from KIIT University Bhubaneswar. His research areas include Object Oriented Software Testing, Software Cost Estimation, and Data mining. Many publications are there to his credit in many International and National level journal and proceedings. He is having eight years of teaching experience. He is a member of ISTE. He can be reached at aacharyafcs@kiit.ac.in.

Durga Prasad Mohapatra received his Master's degree from National Institute of Technology, Rourkela, India. He has received his Ph.D. from Indian Institute of Technology, Kharagpur, India. He is currently working as an Associate Professor at National Institute of Technology, Rourkela. His special fields of interest include Software Engineering, Discrete Mathematical Structure, Program Slicing and Distributed Computing. Many publications are there to his credit in many International and National level journal and proceedings. He is a member of IEEE. He can be reached at durga@nitrkl.ac.in.