BI-GRAPH APPROACH FOR TESTCASE PRIORITIZATION

A PROJECT REPORT

Submitted by

S. MONISHA (810015104048)

K .PONMALAR (810015104060)

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

UNIVERSITY COLLEGE OF ENGINEERING

BIT CAMPUS

TIRUCHIRAPPALLI-620 024

APRIL-2019

UNIVERSITY COLLEGE OF ENGINEERING

BIT CAMPUS

TIRUCHIRAPPALLI-620 024

BONAFIDE CERTIFICATE

Certified that this report titled **"BI-GRAPH APPROACH FOR TESTCASE PRIORITIZATION"** is the bonafide work of Ms. **S. MONISHA(810015104048)** and Ms. **K. PONMALAR (810015104060)** who carried out the work under my supervision. Certified further that to the best of my knowledge the work reported here in does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

Dr.D.VENKATESAN
Assistant Professor &Head
Department of CSE/IT
University College of Engineering
BIT Campus
Tiruchirappalli-620 024

Dr.C.P.INDUMATHI
Assistant Professor
Department of IT
University College of Engineering
BIT Campus
Tiruchirappalli-620 024

Submitted for the VIVA-VOCE examination to be held on………………

Internal Examiner **External Examiner**

DECLARATION

We hereby declare that the work entitled **"BI-GRAPH APPROACH FOR TESTCASE PRIORITIZATION"** is submitted in partial fulfillment of the requirement for the award of the degree in B.E., University College of Engineering, BIT Campus, Anna University, Tiruchirappalli, is record of our own work carried out by us during the academic year 2018-2019 under the supervision and guidance of **Dr.C.P.INDUMATHI**, Assistant Professor , Department of Information Technology, University College of Engineering, BIT Campus, Anna University, Tiruchirappalli. The extent and source of information are derived from the existing literature and have been indicated through the dissertation at the appropriate places. The matter embodied in this work is original and has not been submitted for the award of any degree, either in this or any other University.

S.MONISHA (810015104048)

K.PONMALR (810015104060)

I certify that the declaration made above by the candidates is true.

Signature of the Guide,
Dr.C.P.INDUMATHI,
ASSISTANT PROFESSOR,
Department of IT,
University College of Engineering,
BIT Campus,
Tiruchirappalli-620 024.

ACKNOWLEDGEMENT

It is the great opportunity to express our sincere thanks to all the people who have contributed to the successful completion of our project work through their support encouragement and guidance.

Our first and foremost thanks **to Dr. T. SENTHIL KUMAR**, Dean, University College of Engineering, BIT Campus, Anna University, Tiruchirappalli for their support in doing this project.

It is our privilege to render our sincere thanks to **Dr. D.VENKATESAN**, Head of the department of Computer Science and Engineering, University College of Engineering, BIT Campus, Anna University, Tiruchirappalli for providing us with excellent lab facilitates and ideas.

We wish to record our heartfelt gratitude to our esteemed guide **Dr.C.P.INDUMATHI**, Assistant Professor, Department of Information Technology, University College of Engineering, BIT Campus, Anna University, and Tiruchirappalli for his excellent guidance, enterprising and valuable suggestions, encouragement and inspiration offered throughout the project.

It is our responsibility to thank our project coordinator **Mr. C. SANKAR RAM**, Teaching fellow, Department of Computer Science and Engineering, deserves a special vote of thanks for his constant inspiration that he has been all through the project period. I render my heartfelt thanks to our entire department teaching and non-teaching staff for their enthusiastic encouragement and support throughout this project.

I hearty thank to my friends for helping me in this project directly or indirectly helped me in making this project a complete success.

ABSTRACT

Software maintenance is an important 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. The regression testing, increase the cost and time. This issue of retesting of software systems can be handled using a test case prioritization technique. So, we use test case prioritization technique to reduce the cost and time, improve the performance of the system. The Bi-Graph approach to generate test cases from UML sequence and activity diagrams. First transform these UML diagrams into a graph. Then, propose an algorithm to generate test scenarios from the constructed graph. One potential goal prioritization is to increase a test suite's rate of fault detection. An improved rate of fault detection can provide earlier feedback on the system, enabling earlier debugging. Then, association rule mining (ARM) is applied to the historical data to generate the frequent pattern. Finally, test cases are prioritized based on business criticality .test value (BCTV) and frequent pattern. We have also verified the effectiveness of proposed approach by calculating the percentage of fault detection APFD (Average Percentage of Fault Detected) metric is used to measure the test suite's fault detection rate. In our case study application, Test case prioritization approach to increase the test suite's fault detection rate.

TABLE OF CONTENT

CHAPTER NO	TITLE	PAGE NO
	ABSTRACT	v
	LIST OF TABLES	ix
	LIST OF FIGURES	xi
	LIST OF ABBREVIATIONS	xii
1.	**INTRODUCTION**	**1**
	1.1 REGRESSION TESTING	1
	1.2 TEST CASE SELECTION	2
	1.3 TEST CASE PRIORITIZATION	3
	1.4 TEST SUITE REDUCTION	3
	1.5 MODEL BASED TESTING	4
	1.6 ORGANISATION OF THE CHAPTER	5
2.	**LITERATURE SURVEY**	**6**
	2.1AUTOMATIC TESTCASE GENERATION FROM UML SEQUENCE DIAGRAMS	6
	2.2 A NOVEL APPROACH TO GENERATE TEST CASE FROM UML ACTIVITY DIAGRAMS	8
	2.3 TESTCASE GENERATION FROM UML MODELS	9
	2.4 PRIORITIZING TESTCASES USING BCTV	11
	2.5 A NOVEL APPROACH FOR TESTCASE PRIORITIZATION USING BCTV	13
	2.6 AN APPROACH FOR TESTCASE	14

	PRIORITIZATION BASED ON THREE FACTORS	
	2.7 MODEL BASED TESTCASE PRIORITIZATION USING ARM	16
	2.8 TEST CASE GENERATION FOR CONCURRENT SYSTEM USING UML DIAGRAM	19
	2.9 LIMITATIONS OF EXISTING SYSTEM	21
3.	**SYSTEM ANALYSIS**	**22**
	3.1 DATA FLOW DIAGRAM	22
	3.2 ARCHITECTURE FOR PROPOSED WORK	25
4.	**PROPOSED TECHNIQUE**	**28**
	4.1 TCP USING BCV METHOD	**28**
	4.1.1 ACTIVITY AND SEQUENCE DIAGRAMS	28
	4.1.2 ACTIVITY SEQUENCE GRAPH	31
	4.1.3 GENERATE TEST CASES	33
	4.1.4 FORWARD SLICING ALGORITHM	34
	4.1.5 FIND AFFECTED NODES USING FORWARD SLICING ALGORITHM	36
	4.1.6 GENERATE FPAN USING ARM	36
	4.1.7 GENERATE BCV OF ALL NODES IN FPAN	38

4.1.8 GENERATE PRIORITIZED TEST SCENARIOS USING BCTV 41
4.1.9 FAULT DETECTION 42
4.2 TCP USING WEIGHT METHOD 44
4.2.1 CALCULATE NODE WEIGHT 45
4.2.2 CALCULATE EDGE WEIGHT 46
4.2.3 CALCULATE ACTIVITY SEQUENCE PATH WEIGHT 47
4.2.4 PRIORITIZED ORDER OF TEST CASES 48
4.2.5 FAULT DETECTION 49
5. DISCUSSION ON EXPERIMENTAL RESULTS 51
5.1 EFFECTIVENESS USING APFD METRIC 51
5.2 COMPARATIVE STUDY 53
6. CONCLUSION 55
REFERENCES 56
PUBLICATION DETAILS 60

LIST OF FIGURES

FIGURE NO	TITLE	PAGE NO
1.1	Flow diagram of Regression Testing	2
2.1	Test Case Generation from Sequence Graph	7
2.2	Prioritizing the Test cases Using BCTV	13
2.3	Factors Based Test case prioritization	16
2.4	Test case prioritization using ARM	18
3.1	Context Level Diagram	23
3.2	Level 1 - Generate test case scenarios	24
3.3	Level 2 - Prioritized Test cases	25
3.4	Architecture Diagram for Proposed work	27
4.1.1	Activity Diagram for BBAS	30
4.1.2	Sequence Diagram of BBAS	31
4.1.3	Activity Sequence Graph of BBAS	32
4.1.4	FPAN of BBAS	37
4.1.5	BCV of BBAS	40
4.1.6	APFD values for prioritized and non-prioritized test cases Using BCV method of BBAS	44
4.2.1	APFD values for prioritized and non-prioritized test cases Using Weight method in BBAS	50
5.1	APFD values for different case studies using BCV approach	52

5.2	APFD values for different case studies using Weight approach	53
5.3	Existing vs. Proposed Work –APFD values	54

LIST OF TABLES

TABLE NO	TABLE NAME	PAGE NO
4.1.1	Test Cases for BBAS	34
4.1.2	Modified and Affected Nodes of BBAS	36
4.1.3	BCV for BBAS	40
4.1.4	Prioritized order of test cases in BBAS	41
4.1.5	Total number of faults detected by each test case in BBAS	43
4.2.1	Node Weight in ASG	45
4.2.2	Edge Weight in ASG	47
4.2.3	Total Weight in Test Cases	48
4.2.4	Prioritized order of test cases	49
5.1	APFD values for different case studies using BCV approach	52
5.2	APFD values for different case studies using Weight approach	53
5.3	APFD Values for Existing Vs Proposed work	54

LIST OF ABBREVIATIONS

1.	AD	Activity Diagram
2.	ASG	Activity Sequence Graph
3.	ARM	Association Rule Mining
4.	APFD	Average Percentage of Fault Detection
5.	BBAS	Big Bazaar Automation system
6.	BCV	Business Criticality Value
7.	BCTV	Business Criticality Test Value
8.	CV	Confidence Value
9.	FPAN	Frequent Pattern of Affected Nodes
10.	LMS	Library Management System
11.	MBT	Model Based Testing
12.	MBTP	Model Based Test Case Prioritization
13.	SMMS	Shopping Mall Management System
14.	SV	Support Value
15.	SUT	Software Under Test
16.	SD	Sequence Diagram
17.	TCR	Test Case Ranking
18.	TC	Test Case
19.	TCP	Test Case Prioritization
20.	UML	Unified Modeling Language

CHAPTER-1

INTRODUCTION

1.1 REGRESSION TESTING

Regression Testing is defined as a type of software testing to confirm that a recent program or code change has not adversely affected existing features. Regression Testing is a full or partial selection of already executed test cases which are re-executed to ensure existing functionalities work fine. This testing is done to make sure that new code changes should not have side effects on the existing functionalities. It ensures that the old code still works once the new code changes are done. 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**.** The main aim of regression testing is to test the modified software during maintenance level. It is an expensive activity, and it assures that modifications performed in software are correct. An easiest strategy to regression testing is to re-test all test cases in a test suite, but due to limitation of resource and time, it is inefficient to implement. Therefore, it is necessary to discover the techniques with the goal of increasing the regression testing's effectiveness, by arranging test cases of test suites according to some objective criteria. Regression testing is a type of software testing, aims to validate enhanced software, and it confirms that all the modifications done on software are correct. Regression Testing is required when there is a Change in requirements and code is modified according to the requirement, new feature is added to the software, fix the defect and performance issue. Software maintenance is an activity which includes enhancements, error corrections, optimization and deletion of existing features. These modifications may cause the system to work incorrectly.

Therefore, Regression Testing becomes necessary. Flow diagram of Regression Testing is shown in the figure 1.1.

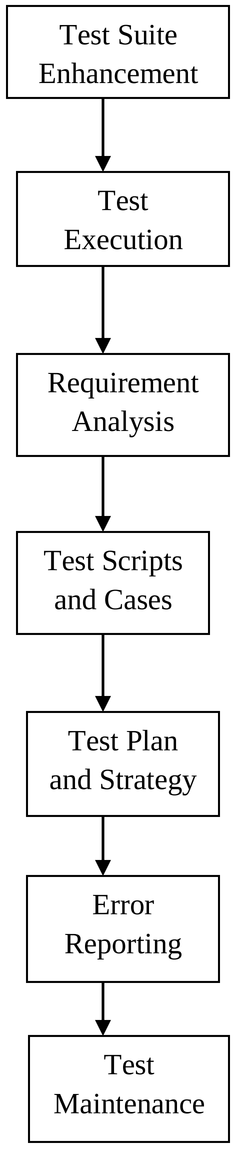


Fig 1.1 Flow diagram of Regression Testing

1.2 TEST CASE SELECTION

One of the main types of regression testing, which is also considered its biggest field is Test Case Selection. The concept of Test Case Selection comes in tandem with the definition of Regression Testing. As elaborated earlier it is both expensive and impractical to simply retest all the test cases after software or program has updated. Thus a method is required in order to identify and select these test cases that are relevant based on a particular criterion.

1.3 TEST CASE PRIORITIZATION

Test Case Prioritization is an essential part of Regression Testing. Test Case Prioritization first and foremost objective is to increase fault detection rate which is defined as the rate at which a test suit can find faults within the testing process. As such the impact of this process can offers and earlier response from the system as it is being tested, what this implies is that when or if the testing process is halted for whatever reason, due to early debugging the prioritization has enabled the execution of the most critical test cases at an earlier stage . In essence the test cases are prioritized according to their priority which is determined based on various criteria depending on the technique used, and executed according to the order the test cases are prioritized. Test Case Prioritization is a methodology used for scheduling of test cases with the intention of detecting maximum faults with minimum time and cost. Prioritization also focuses on increase in rate of high risk faults detection and ensures reliability of software products at a faster rate. Researchers had proposed several criteria for scheduling of test cases during prioritization such as, coverage-based, requirement-based, risk factor based etc. But, test cases with maximum coverage do not confirm maximum fault detection. Hence we have to propose a technique which should give maximum coverage with high fault detection capability.

1.4 TEST SUITE REDUCTION

As the software evolves during the process of software development, new test cases have to be written in order to validate the newly added features which in many scenarios also create new faults that need to be tested. As new test cases are created to better compliment the new test requirements, there will be instances that a single test requirement will be represented by more than a single test case. This creates a form of redundancy with test cases covering a single test requirement. Test Suit Reduction or Minimization is the process of reducing and removing the redundant test cases.

1.5 MODEL BASED TESTING

Model-based testing is an application of model-based design for designing and optionally also executing artifacts to perform software testing or system testing. Models can be used to represent the desired behavior of a system under test, or to represent testing strategies and a test environment. Test cases derived from such a model are functional tests on the same level of abstraction as the model. These test cases are collectively known as an abstract test suite. An abstract test suite cannot be directly executed against an SUT because the suite is on the wrong level of abstraction. An executable test suite needs to be derived from a corresponding abstract test suite. Tests can be derived from models in different ways. Because testing is usually experimental and based on heuristics, there is no known single best approach for test derivation. In the proposed approach, UML activity sequence diagram is used as system model and test case Prioritization is performed by analyzing the modification history of different version of that SUT. First the system model is converted to a model dependency graph called Activity Sequence Graph (ASG) and test cases are generated from ASG for the new version of SUT. Then, the proposed forward slicing algorithm is used on the ASG to track the modified node and trace out the affected nodes of all modified nodes.

1.6 ORGANIZATION OF THE CHAPTER

Chapter are Organized are as follows. Chapter 2 describes the literature survey of the existing papers. Chapter 3 discusses the data flow diagram and its levels and system Architecture. Chapter 4 describes the proposed technique and experimentation and analysis of the given project. Chapter 5 describes the Discussion on Experimental results. Chapter 6 describes the Conclusion and Future work.

CHAPTER-2

LITERATURE SURVEY

2.1 AUTOMATIC TESTCASE GENERATION FROM UML SEQUENCE DIAGRAMS

The paper [2] defines a novel approach of generating test cases from UML design diagrams. Automatic Test Case generation consists of transforming a UML sequence diagram into a graph called the sequence diagram graph (SDG) and augmenting the SDG nodes with different information necessary to compose test vectors. These information are mined from use case templates, class diagrams and data dictionary. The SDG is then traversed to generate test cases. The test cases thus generated are suitable for system testing and to detect interaction and scenario faults.

One significant approach is the generation of test cases from UML models. The main advantage with this approach is that it can address the challenges posed by object-oriented paradigms. Moreover, test cases can be generated early in the development process and thus it helps in finding out many problems in design if any even before the program is implemented. However, selection of test cases from UML model is one of the most challenging tasks. A test case consists of a test input values, its expected output and the constraints, that is the pre- and post condition for that input values. This information may not be readily available in the Design artifacts. As a way out to this problem, several researches propose to augment the design models with testable information prior to the testing process. However, this complicates the automatic test case generation effort.

The automatic test case generation method used for UML models. Sequence diagram as a source of test case generation. This generated test suite aims to cover various interaction faults as well as scenario faults. For generating test data, sequence diagram alone may not be enough to decide the different components, i.e. input, expected output and pre- and post- condition of a test case.

Given a sequence diagram (SD), and transform it into a graphical representation called sequence diagram graph (*SDG*). Each node in the *SDG* stores necessary information for test case generation. This information are collected from the use case template (also called extended use case), class diagrams, and data dictionary expressed in the form of object constrained language (OCL), which are associated with the use case for which the sequence diagram is considered. Then traverse *SDG* and generate test cases based on a coverage criteria and a fault model. This methodology has been shown in fig 2.1.

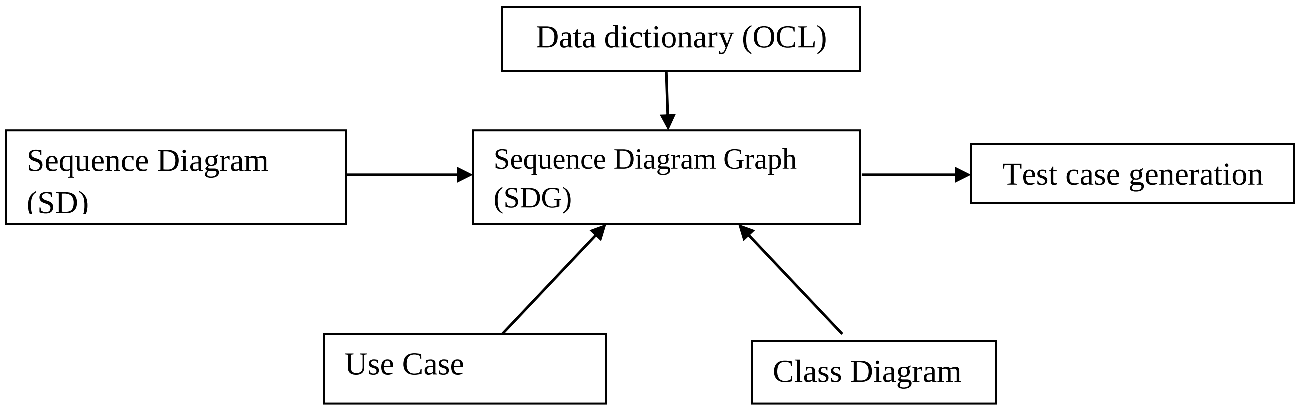


Fig 2.1 Test Case Generation from Sequence Graph

A methodology has been proposed to convert the UML sequence diagram into a graph called sequence diagram graph. The information those are required for the specification of input, output, pre- and post- conditions etc. of a test case are Retrieved from the extended use cases, data dictionary expressed in OCL 2.0, class diagrams (composed of application domain classes and their contracts) etc. and are stored in the *SDG*. The approach does not require any modification in the UML models or manual intervention to set input/output etc. to compute test cases. Hence, this approach provides a tool that straightway can be used to automate testing process. Use a graph based methodology and run-time complexity is governed by the breadth-first search algorithm to enumerate all paths, which is *O(n2)*in the worst case for a graph of *n* nodes. This implies that proposed approach can handle a large design efficiently.

2.2 A NOVEL APPROACH TO GENERATE TEST CASE FROM UML ACTIVITY DIAGRAMS

The paper [3] defines Model-based test case generation is gaining acceptance to the software practitioners. Advantages of this are the early detection of faults, reducing software development time etc. In recent times, researchers have considered different UML diagrams for generating test cases. Few works on the test case generation using activity diagrams is reported in literatures. However, the existing work considers activity diagrams in method scope and mainly follows UML 1.*x* for modeling. In this paper, present an approach of generating test cases from activity diagrams using UML 2.0 syntax and with use case scope. Consider a test coverage criterion, called activity path coverage criterion. The test cases generated using our approaches are capable of detecting more faults like synchronization faults, loop faults unlike the existing approaches.

Model-driven software development is a new software development paradigm. Its advantages are the increased productivity with support for visualizing domains like business domain, problem domain, solution domain and generation of implementation artifacts. In the model-driven software development, practitioners also use the design model for testing software- especially object-oriented programs. Three main reasons for using design model in object-oriented program testing are: (1) traditional software testing techniques consider only static view of code which is not sufficient for testing dynamic behavior of object-oriented system .(2) use of code to test an object-oriented system is complex and tedious task. In contrast, models help software testers to understand systems better way and find test information only after simple processing of models compared to code, (3) model-based test case generation can be planned at an early stage of the software development life cycle, allowing to carry out coding and testing in parallel. For these three major reasons, model-based test case generation methodology becomes an obvious choice in software industries and is the focus of this paper.

Activity diagram is an important diagram among 13 diagrams supported by UML 2.0. It is used for business modeling, control and object flow modeling, complex operation modeling etc. Main advantage of this model is its simplicity and Ease of understanding the flow of logic of the system. However, finding test information from activity diagram is a formidable task.

To generate test cases from an activity diagram consists of the following three steps.

1. Augmenting the activity diagram with necessary test information.
2. Converting the activity diagram into an activity graph.
3. Generating test cases from the activity graph.

In this method, Generating test cases from activity diagram at use case scope. This approach is significant due to the following reasons. First, this is capable to detect more faults like faults in loop, synchronization faults than the existing approaches. Second, test case generated in this approach may help to identify location of a fault in the implementation, thus reducing testing effort. Third, the model-based test case generation approach in-spires developer to improve design quality, find faults in the implementation early, and reduce software development time. Fourth, it is possible to build an automatic tool following approach. This automatic tool will reduce cost of software development and improve quality of the software.

2.3 TESTCASE GENERATION FROM UML MODELS

The integrated approach for generating, the test cases from UML sequence and activity diagrams. First transform these UML diagrams into a graph. Then, we propose an algorithm to generate test scenarios from the constructed graph. Next, the necessary information for test case generation, such as method-activity sequence, associated objects, and constraint conditions are extracted from test scenario. This approach reduces the number of test cases and still achieves adequate test coverage.

UML models are an important source of information for test case design, which if satisfactorily exploited, can go a long way in reducing testing cost and effort and at the same time improve software quality .UML-based automatic test generation is a practically important and theoretically challenging topic and is receiving increasing attention from researchers. Traditionally there have been lots of efforts to generate test cases from UML diagrams using heuristic based techniques such as statement-coverage, branch-coverage, message sequence coverage etc.

The test generation process is divided into three main phases. The first phase is to generate MFG from sequence and activity diagram separately. The second phase is to generate test sequences from MFG corresponding to sequence and activity diagrams. The test sequences are a set of theoretical paths starting from initialization to end, while taking conditions (pre-condition and post-condition) into consideration. Each generated test sequence corresponds to a particular scenario of the considered use case. The third phase is to generate test case from the generated sequences satisfying the message-activity path test adequacy criteria.

To generate test cases automatically from UML sequence and activity. Then convert the models into an intermediate representation called Model diagrams Flow Graph (MFG), which is an integration of intermediate representation of sequence and activity diagrams. Integration of these representations is helpful for the following reasons. This approach covers three important faults, which usually occur in a system: message sequence faults, operation consistency faults and activity synchronization faults. The first two category of faults can be covered from the sequence diagram, whereas the later from the activity diagram. The proposed approach is meant for cluster level testing where object interactions are tested. It may be noted that MFG models the operational details of a use case. The integration will help us to guide whether a test driver needs to apply a specific test suite or not. Another, important reason of integrating is that test data those are necessary for test case are mined once and used in different level such as, sequence diagram (message sequence faults within the logic of one operation), activity diagram (to test activity synchronization fault) etc. Integration of models also uncovers new sequence of message activity faults. In this paper, Tests are intended to exercise behavioral paths determined by conditions and uncover faults related to interactions between objects..

2.4 PRIORITIZING TESTCASES USING BCTV

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 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 prioritize the test cases.

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 un minimized one, other empirical evidence shows that the fault detection capabilities of test suites can be severely compromised by minimization. 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. The Architecture of TCP using BCTV shown in Fig 2.2.

To proposed 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.

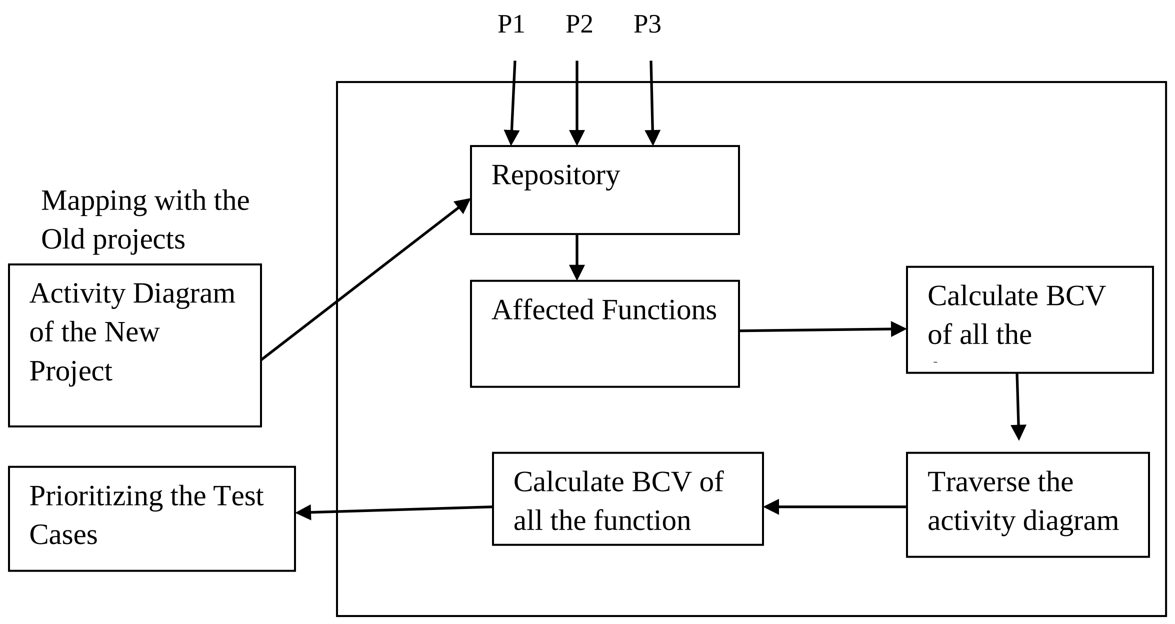

P1
P2
P3
Repository
Mapping with the Old projects
Activity Diagram of the New Project
Affected Functions
Calculate BCV of all the
Prioritizing the Test Cases
Calculate BCV of all the function
Traverse the activity diagram

Fig 2.2 Prioritizing the Test cases Using BCTV

This approach is a model-based test case prioritization which is specifically contains the functional features of the project

2.5 A NOVEL APPROACH FOR TESTCASE PRIORITIZATION USING BCTV

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 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.

In this section, three approaches are used in prioritizing the test cases.

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.

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 method is completely model-based. In this method 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.

2.6 AN APPROACH FOR TESTCASE PRIORITIZATION BASED ON THREE FACTORS

The main aim of regression testing is to test the modified software during maintenance level. It is an expensive activity, and it assures that modifications performed in software are correct. An easiest strategy to regression testing is to

Re test all test cases in a test suite, but due to limitation of resource and time, it is inefficient to implement. Therefore, it is necessary to discover the techniques with the goal of increasing the regression testing's effectiveness, by arranging test cases of test suites according to some objective criteria. Test case prioritization intends to arrange test cases in such a manner that higher priority test cases execute earlier than test cases of lower priority according to some performance criteria.

To prioritize regression test cases based on three factors which are rate of fault detection, percentage of fault detected and risk detection ability. This is compared with different prioritization techniques such as no prioritization, reverse prioritization, random prioritization. For prioritization, test cases are arranged in decreasing order of test case ranking value. Test cases are arranged in such a way that those with greater test case ranking values executes earlier.

The factors are:

- **Rate of Fault Detection**

  The rate of fault detection (RFT) is defined as the average number of defects found per minute by a test case.

- **Percentage of Fault Detected**

  The percentage of fault detected (PFD) for test case Tj can be computed by using number of faults found by test case Tj and total number of faults.

- **Risk Detection Ability**

  It can be defined as the ability of test case to detect severe faults per unit time. Testing efficacy could be progressed by emphasizing on test cases which detect Greater percentage of severe faults.

The algorithm to prioritize test cases based on three factors which are rate of fault detection, percentage of fault detected and risk detection ability is proposed. Testing efficacy could be progressed by emphasizing on test cases which detect greater percentage of severe faults. For every test case all the three factors are calculated and test case ranking is computed by adding these factors for each test case. To solve the problem of test case prioritization prioritizes test cases, according to decreasing order of test case ranking value, and obtains the prioritized order of test cases. This is compared with different prioritization techniques such as no ordering, reverse prioritization, random prioritization, and using APFD metric. The APFD is calculated by taking the weighted average of the number of faults detected during the execution of the test suite. Diagrammatic Representation of Test case prioritization shown in Fig 2.3.

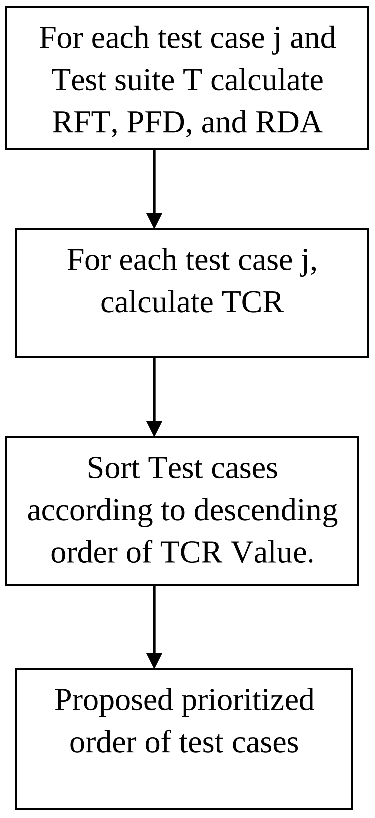

For each test case j and Test suite T calculate RFT, PFD, and RDA
For each test case j, calculate TCR
Sort Test cases according to descending order of TCR Value.
Proposed prioritized order of test cases

Fig 2.3 Factors Based Test case prioritization

2.7 MODEL BASED TESTCASE PRIORITIZATION USING ARM

Regression testing has gained importance due to increase in frequency of change requests made for software during maintenance phase. The retesting criteria of regression testing leads to increasing cost and time. Prioritization is an important procedure during regression testing which makes the debugging easier. The method is used for novel approach for test case prioritization using Association Rule Mining (ARM). The system under test is modeled using UML Activity Diagram (AD) which is further converted into an Activity Graph (AG). A historical data store is maintained to keep details about the system which revealing more number of faults. Whenever a change is made in the system, the frequent Patterns of highly affected nodes are found out. These frequent patterns reveal the Probable affected nodes i.e. used to prioritize the test cases. This approach effectively prioritizes the test cases with a higher Average Percentage of Fault Detection (APFD) value.

A new heuristic approach for test case prioritization is presented. In the proposed approach, UML activity diagram is used as system model and test case prioritization is performed by analyzing the modification history of different version of that SUT. First the system model is converted to a model dependency graph called Activity Graph (AG) and test cases are generated from AG for the new version of SUT. Then, the proposed forward slicing algorithm is used on the AG to track the modified node and trace out the affected nodes of all modified nodes.

Simultaneously historical data of affected nodes of the previous version of SUT are analyzed. The historical data contains Graph Data (GD) and Observation Data (OD). GD and OD for each modified node are used to find out a common pattern of affected nodes revealing more number of faults. Then, test case prioritization is done by using the patterns of affected nodes generated using ARM. Frequent patterns generated using association rule mining helped effectively in prioritization. This approach gives a better result with respect to the average percentage of fault detection. Test case prioritization using ARM can be shown in the figure 2.4.

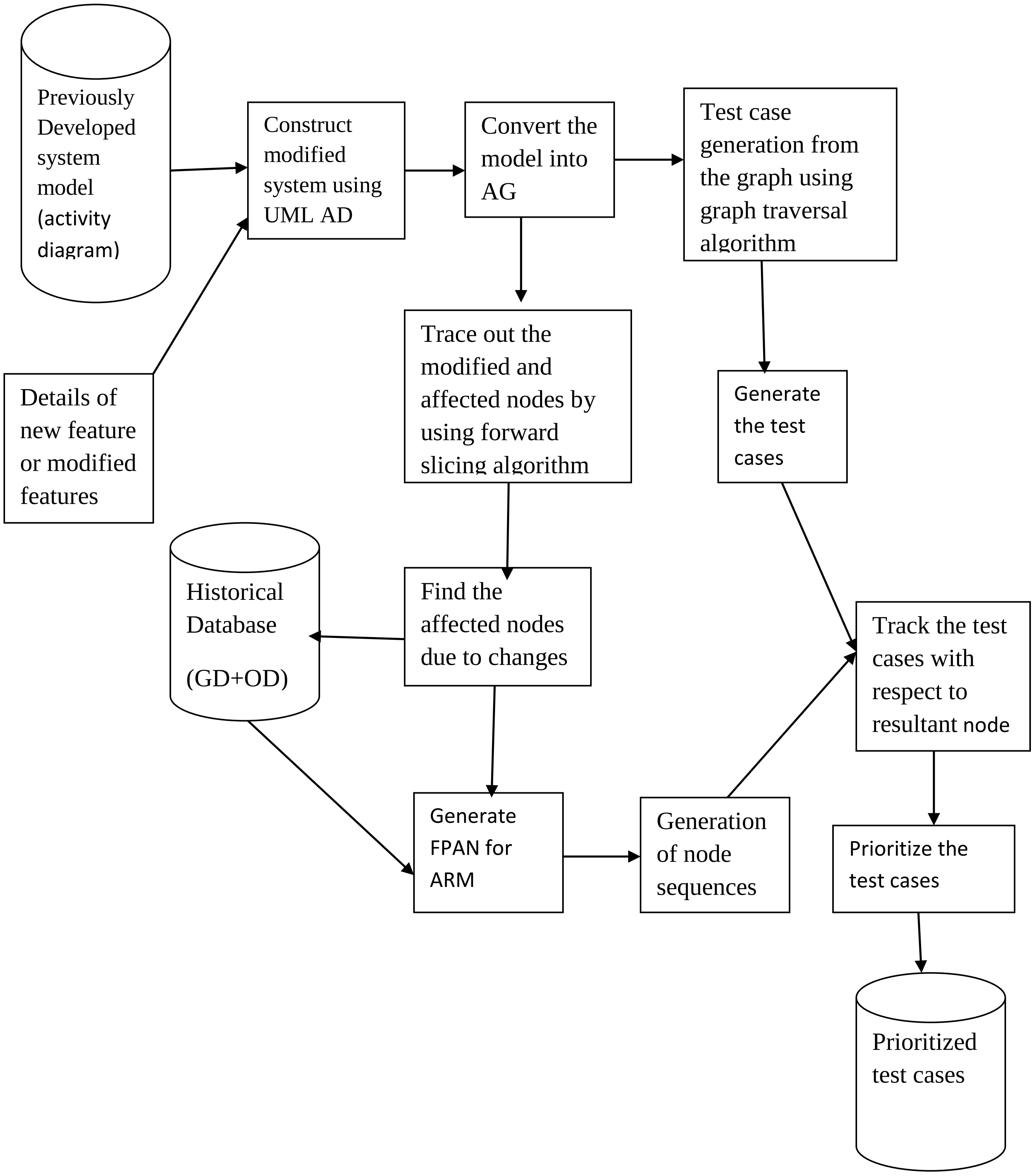

Previously Developed system model (activity diagram)
Construct modified system using UML AD
Convert the model into AG
Test case generation from the graph using graph traversal algorithm
Details of new feature or modified features
Trace out the modified and affected nodes by using forward slicing algorithm
Generate the test cases
Historical Database (GD+OD)
Find the affected nodes due to changes
Track the test cases with respect to resultant node
Generate FPAN for ARM
Generation of node sequences
Prioritize the test cases
Prioritized test cases

Fig 2.4 Test case prioritization using ARM

2.8 TEST CASE GENERATION FOR CONCURRENT SYSTEM USING UML DIAGRAM

The unreasonable interference of concurrent threads makes the testing activity for concurrent systems a difficult task. Test case explosion is the major problem in concurrency testing and make an interruption in systematic testing of concurrent systems. In this method to generate test cases from combinational UML models. In this approach Activity Diagram (AD) and Sequence Diagram (SD) are used to model a system. The AD has converted into a graph called Activity Graph (AG) and SD into a graph called Sequence Graph (SG). Finally AG and SG are combined to form a graph called Activity Sequence Graph (ASG). The ASG is traversed using a traversing algorithm to generate the test cases. After comparing the test cases generated from ASG with the test cases generated from AG and SG, it is found that the test cases generated from ASG gives a better coverage when compared with the test cases from single modeling graph. The test cases are generated by controlling the test case explosion and are useful for controlling synchronization fault, loop fault, as well as scenario faults and interaction faults.

In this method the SD available in UML 2.0. The UML Sequence Diagram, also known as interaction diagram, represents the scenarios as possible sequence of message exchange between the objects to specify the task. The Sequence Diagram available in UML 2.0 enables complex scenario to be specified in a single Sequence Diagram. UML 2.0 combines multiple scenarios by means of **Combined Fragment (CF).** A CF may contain another CF, this features allows complex scenarios to be specified in a single SD. A CF encloses one or more processing sequences in a frame which are executed under specific fragment operator. There are 12 different type of fragment operator, but we will be discussing only those operators which have used in proposed works.

Combined Fragment (Par): Typically, the interaction fragment *par* denotes the parallel merge among the messages in the operands of a par fragment.

Combined Fragment Alt*:* The fragment *alt*, denote a choice of behaviors, which to be controlled by an interaction constraint.

In this method to generating test cases from UML combinational diagram i.e. Activity Diagram (AD) and Sequence Diagram (SD). Converted the AD and the SD into intermediate formats called Activity Graph (AG), Sequence Graph (SG) respectfully. Finally combined the AG and the SG to form a combined graph called Activity Sequence Graph (ASG) and traversed the ASG to generate the test cases. The resultant test cases show that the test cases generated from the ASG is having more fault detection capabilities than the single modeling graphs.

Consider the last test sequence obtained from the Sequence Diagram. Here only find information about the message passed between the objects, but can't get any information about how the activity flow occurs. So this test sequence will be capable of detecting faults associated with message sequencing, and will not able to detect fault associated with decision node or loop faults. For example whenever insert a card then suppose the last test sequence is obtained, and then it is a valid test sequence for valid card and password. So this is unable to detect the faults associated with decision. Now consider the last test sequence obtained from Activity Diagram. Then find out the faults associated with decision as well as faults. For example whenever the card is verified then output of the decision node invalid IC or valid IC is represented in the decision thread. But we are not able to find out the messages passed between the objects. On the other hand consider that the last test sequence of the Activity Sequence Graph, here the activities as well as message sequence is considered. So we will be able to detect more number of faults.

2.9 LIMITATIONS OF EXISTING SYSTEM

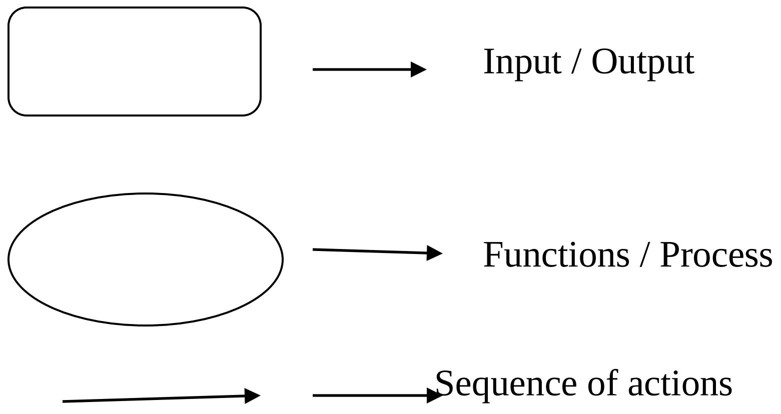
There are several techniques available for prioritizing and reducing the test cases. Some of the test case prioritization techniques are explained and there are certain limitations for intend to apply requirement changes for performing the prioritization techniques and also considering varied factors like cost, efficiency and performance for testing. The test cases are prioritized and reduced effectively so that cost will be effective. Test case prioritization method is more effective and efficient. We proposed, Bi-graph approach, Combining UML Diagrams for Activity and Sequence Diagrams for test case prioritization to improve rate of fault detection than existing systems.

CHAPTER 3
SYSTEM ANALYSIS

3.1 DATA FLOW DIAGRAM

A data flow diagram (DFD) maps out the flow of information for any process or system. It uses defined symbols like rectangles, circles and arrows, plus short text labels, to show data inputs, outputs, storage points and the routes between each destination. A Data Flow Diagram (DFD) is a graphical representation of the “flow” of data through an information system, modeling its process aspects. A DFD is often used as preliminary step to create an overview of the system which can later can be elaborated. DFD can also be used for visualization of data processing (Structure design).

A DFD Shows what kind of information will be input to output from the system, where the data will come from and go to, and where the data will be store. It doesn’t show information about the timing of process or information about whether processors will operate in sequence or parallel.



Level 0

DFD Level 0 is also called a Context Diagram. It’s a basic overview of the whole system or process being analyzed or modeled. It’s designed to be an at-aglance view, showing the system as a single high-level process, with its relationship to external entities. It should be easily understood by a wide audience, including stakeholders, business analysts, data analysts and developers.

At the beginning the activity and sequence diagrams as a input. Combining the activity and sequence diagrams to generate the activity sequence graph. Using depth first search to traverse the graph to generating the test cases. Finally generate the prioritized test cases. The Context level diagram is shown in fig 3.1.

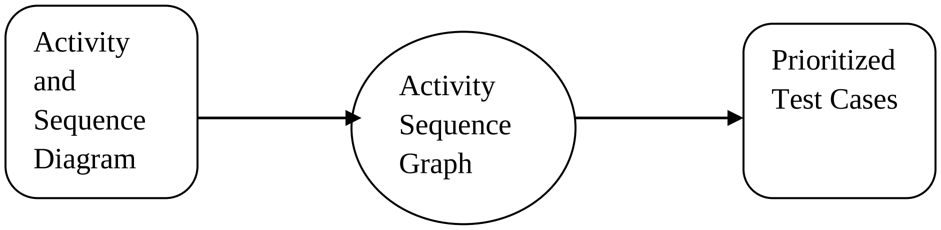


Fig 3.1 Context Level Diagram

Level 1

DFD Level 1 provides a more detailed breakout of pieces of the Context Level Diagram. You will highlight the main functions carried out by the system, as you break down the high-level process of the Context Diagram into its sub processes. Activity and Sequence Diagram as a input. Test cases will be generated in Activity Sequence Graph. To calculate Frequent Pattern Affected Nodes in each test case. Generation of frequent pattern for affected node using ARM. Trace out all modified and affected nodes using Forward Slicing Algorithm. To generating prioritized test cases. Level 1 diagram shown in Fig 3.2.

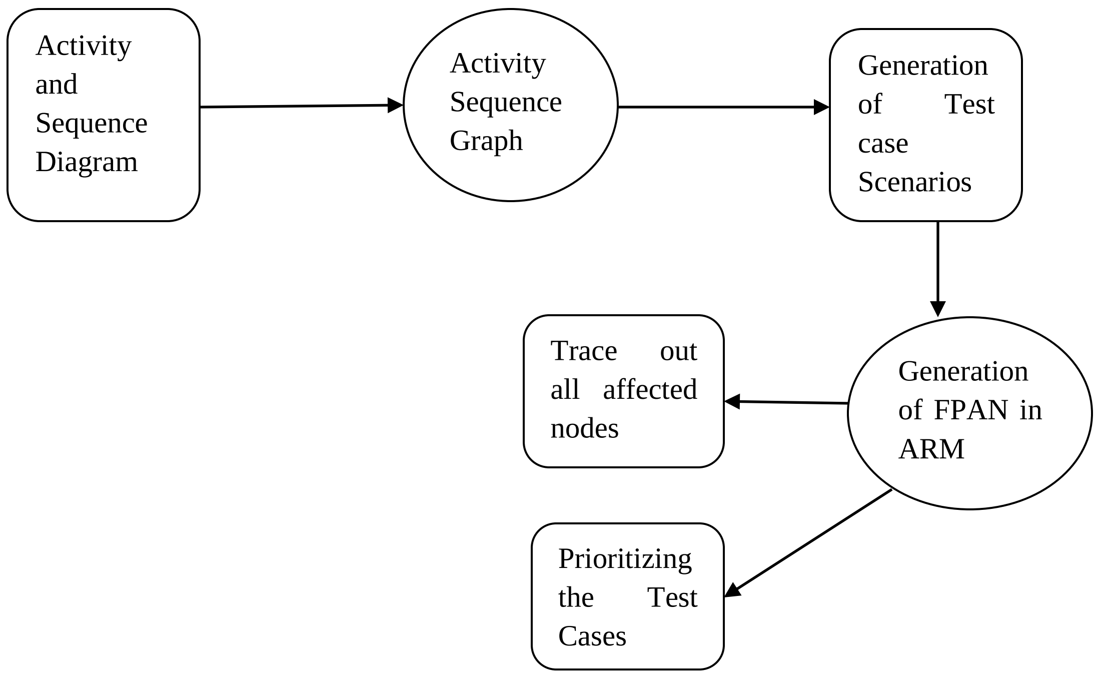

Activity and Sequence Diagram
Activity Sequence Graph
Generation of Test case Scenarios
Generation of FPAN in ARM
Trace out all affected nodes
Prioritizing the Test Cases

Fig 3.2 Level 1 - Generate test case scenarios

Level 2

DFD Level 2 then goes one step deeper into parts of Level 1. It may require more text to reach the necessary level of detail about the system's functioning. Activity and Sequence Diagram as a input. Test cases will be generated in Activity Sequence Graph. To Generate the Frequent Pattern Affected Nodes in each test case. Generation of frequent pattern for affected node using ARM. Trace out all modified and affected nodes using Forward Slicing Algorithm. To Generate BCV of All nodes. To Generate BCTV of all test cases. Finally to generate prioritizing the test cases. The complete the steps of dataflow diagram shown in Fig 3.3.

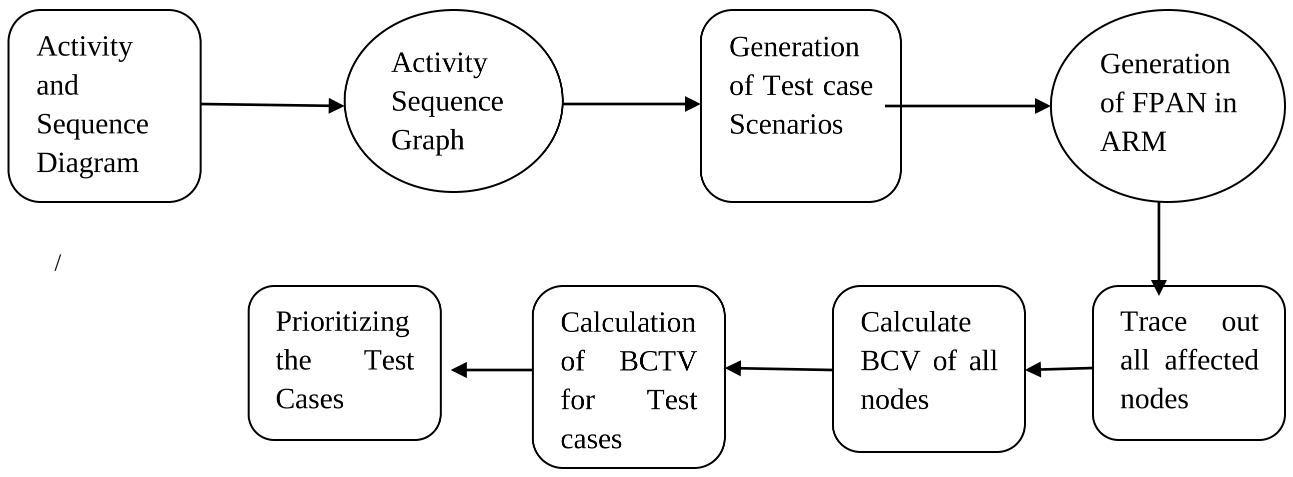

Activity and Sequence Diagram
Activity Sequence Graph
Generation of Test case Scenarios
Generation of FPAN in ARM
Trace out all affected nodes
Calculate BCV of all nodes
Calculation of BCTV for Test cases
Prioritizing the Test Cases
/

Fig 3.3 Level 2 - Prioritized Test cases

3.2 ARCHITECTURE FOR PROPOSED WORK

Consider the UML Model Diagram Such as Activity Diagram & Sequence Diagrams are input for proposed work. In this proposed work, we have taken AD and SD as system model and an model dependency graph named ASG is developed from these two diagrams. ASG is developed by combining behavioural Features and functionalities of both the diagrams. Then, the ASG is traversed from source to destination to generate linearly independent paths and that paths are considered as test cases / scenarios. We have used the terms ‘‘test case’’ and ‘‘test scenario’’. Simultaneously, The details of the graph is stored in a database named Project Repository. The project repository comprises the detail information of the project like Project ID, Total number of Nodes, Modified Nodes and Corresponding Affected Nodes, Test cases, business criticality value of nodes and business criticality test value. After that, we generate a frequent pattern called FPAN using association rule mining (ARM). For frequent pattern generation, all the modified nodes and associated affected nodes are imported as input and we get a pattern consisting of nodes, as the result. Here, we have used a forward slicing algorithm to find out the affected nodes for particular modified node. Then, these nodes are imported to ARM technique. To generate a frequent pattern using ARM, two parameters are required i.e. support value and confidence value. First, the SV of individual items (i.e. nodes) and combination of items are determined. Then, the final value is compared with the minimum or threshold support value and the derived item set is known as frequent item set. Similarly, the CV of each frequent item set is determined and also compared with threshold CV. Then, the generated final pattern is known as Frequent Pattern of Affected Node (FPAN). To produce the frequent pattern by taking minimum support value as 40% and minimum confidence value as 80%.In the last step, test cases are prioritized by using BCTV. BCTV of test case is determined by taking the summation of BCV of all nodes executed by the test case. The TC containing highest BCTV is considered as highest priority TC and the TC with lowest BCTV is considered as lowest priority TC. The test cases are prioritized based on the priority of BCTV of TCs. Lastly, the prioritized test suite is stored in the project repository for subsequent implementation.

The Architecture for our proposed work is given as follows in fig 3.4.

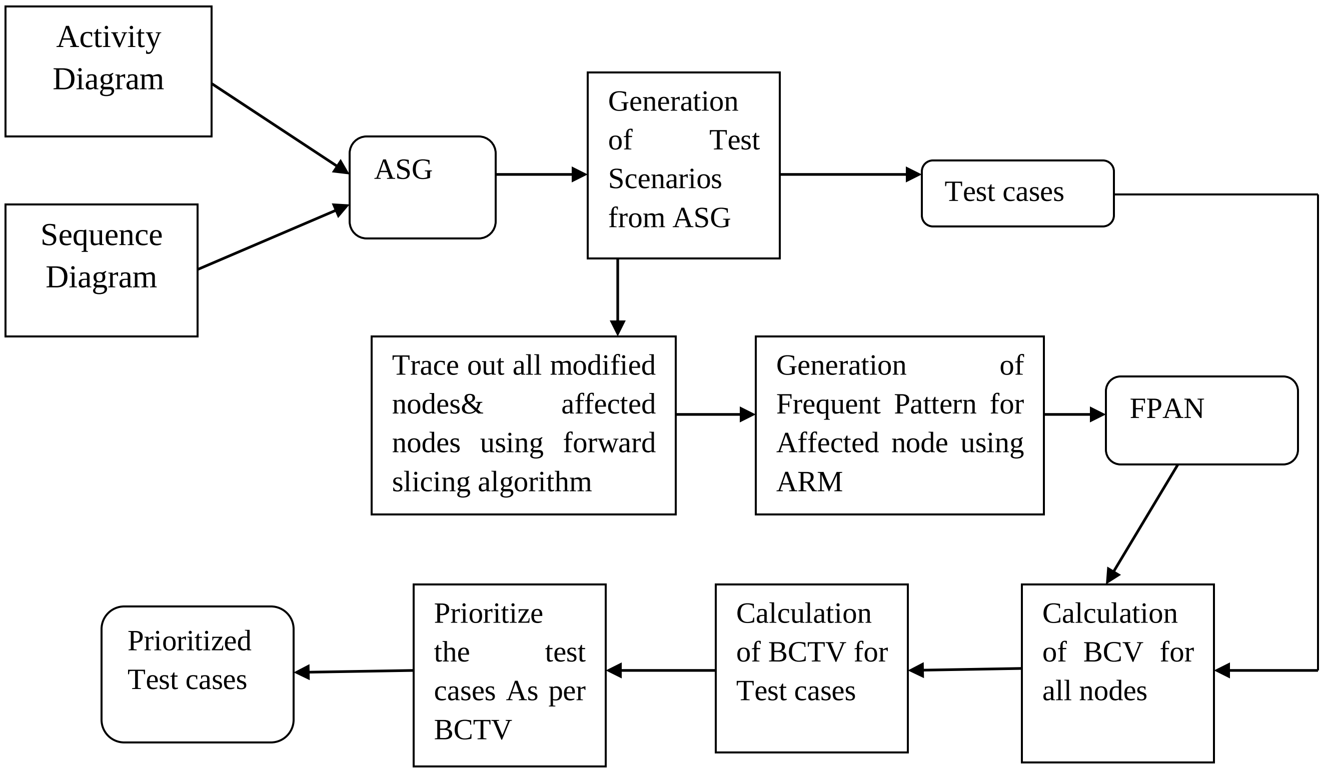

Activity Diagram
Sequence Diagram
ASG
Generation of Test Scenarios from ASG
Test cases
Trace out all modified nodes& affected nodes using forward slicing algorithm
Generation of Frequent Pattern for Affected node using ARM
FPAN
Calculation of BCV for all nodes
Calculation of BCTV for Test cases
Prioritize the test cases As per BCTV
Prioritized Test cases

Fig 3.4 Architecture Diagram for Proposed work

CHAPTER 4

PROPOSED TECHNIQUE

To prioritize the test cases we have proposed two approaches one is BCV value approach and another one is Weight value approach.

4.1 TCP USING BCV METHOD

In BCV value approach

- ➢ We have taken AD and SD as system model.
- ➢ ASG is developed from these two AD and SD
- ➢ ASG is traversed from source to destination to generate test case
- ➢ Using Forward Slicing Algorithm find affected nodes for the respective changes.
- ➢ After that, generate FPAN using ARM.
- ➢ Calculate BCV of all FPAN.
- ➢ Calculate BCTV of all test cases using BCV values.
- ➢ In last step, test cases are prioritized by using BCTV.

The detail procedure of proposed approach using Bi-graph is given below:

4.1.1 ACTIVITY and SEQUENCE DIAGRAMS

In the first step, we collect the modified requirements and model the system with new requirements. UML (Unified Modeling Language) behavioural diagrams such as activity diagram and sequence diagram are used to model the system. In most of the companies, software developers prefer Unified Modeling Language (UML) diagrams to design the system model, because UML diagrams are easy to visualize, design and document. In this work, we have also considered UML sequence diagram and activity diagram as input. Sequence diagrams capture the exchange of messages between objects during execution of a use case. It focuses on the order in which the messages are sent. Activity diagrams, on the other hand, focus upon control flow as well as the activity-based relationships among objects.

These are very useful for visualizing the way several objects collaborate to get a job done. These are very useful for describing the procedural flow of control through many objects.

To explain the detail functioning of the proposed framework, we have used a case study of Library Management System (LMS), Shopping Mall Management System (SMMS) and Big Bazaar Automation System (BBAS). Here we explain Big Bazaar Automation System is a software which automates the activities in a big bazaar such as choose product, billing, swap payback card, payment mode, more option, take receipt, etc. The activity diagram for BBAS is shown in Fig 4.1.1 Sequence diagram for BBAS shown in Fig 4.1.2 that contains sequence of messages and roles. The roles are Customer, Billing section, payment Section, and Other services. And the messages are Choose product, swap payback card, enter pin, more option, etc.

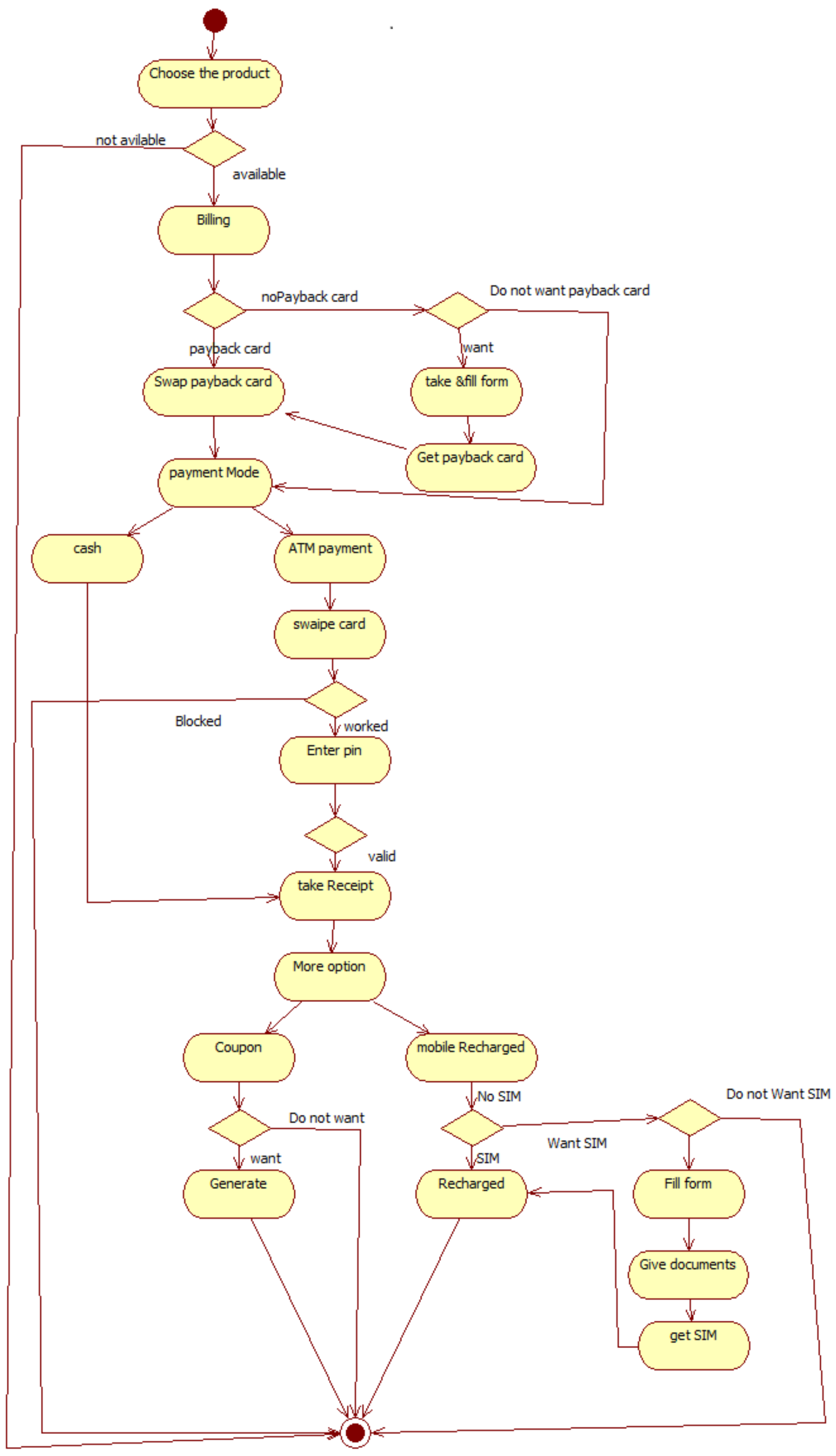


Fig 4.1.1 Activity Diagram for BBAS

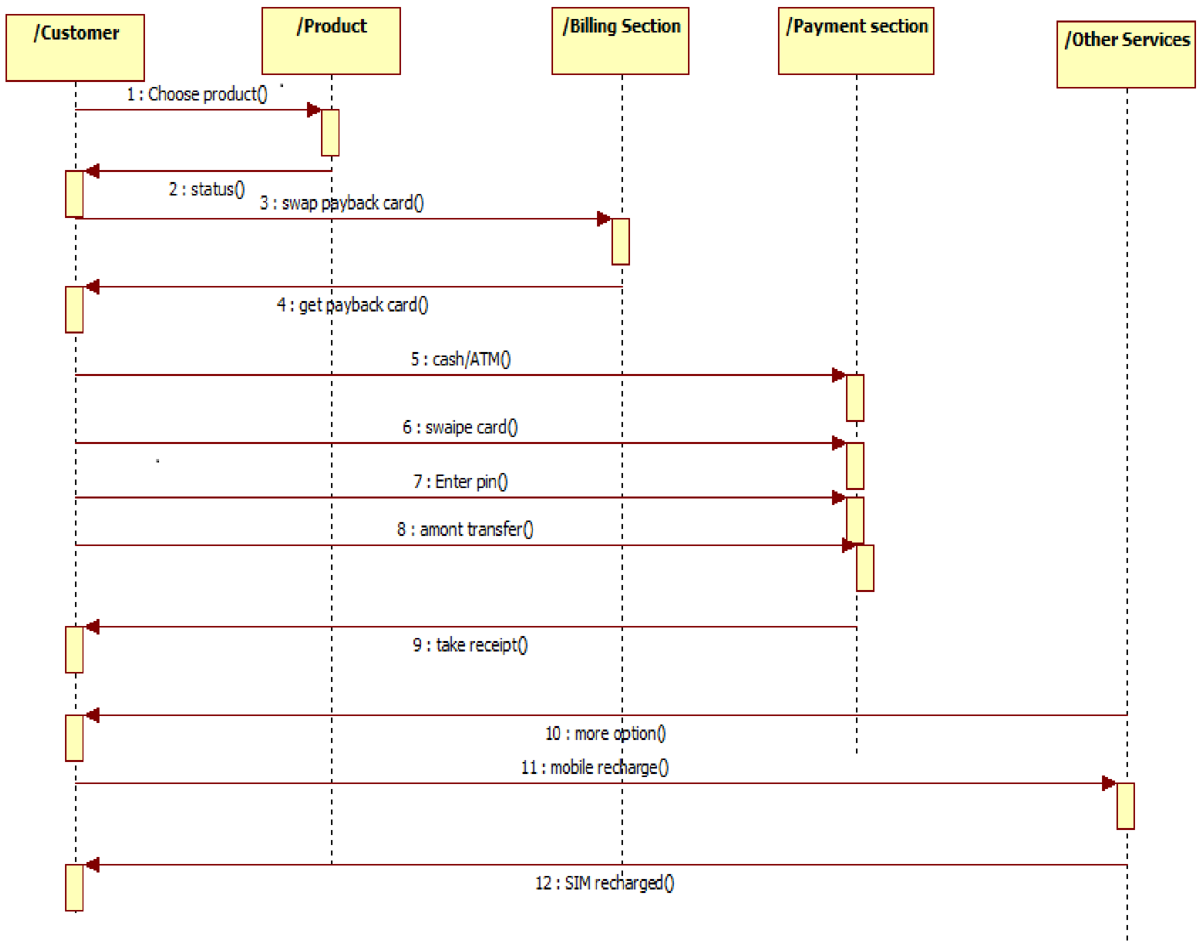


Fig 4.1.2 Sequence Diagram of BBAS

4.1.2 ACTIVITY SEQUENCE GRAPH

To transform the activity and sequence diagrams are converted into intermediate graph. From the constructed graph, we generate different test sequences, which represent different scenarios. From the generated test sequences, test cases are generated, which satisfy the message sequence test path adequacy criteria. We focus on generating tests from design description, as it represents a significant opportunity for testing in a form that can easily be manipulated by automated means. Collect the related features of Activity Diagram (AD) and Sequence Diagram (SD). Consider the similar feature as a single activity and construct an intermediate graph named Activity Sequence Graph (ASG). During the conversion, each node in the ASG represents the activity information along with the message transmitted related to that activity. The connectors between the activities (i.e. transitions or edges in AD) expressed as edges in the activity sequence graph. Then, this graph is traversed to generate the test scenarios. The obtained ASG is shown in Fig 4.1.3.

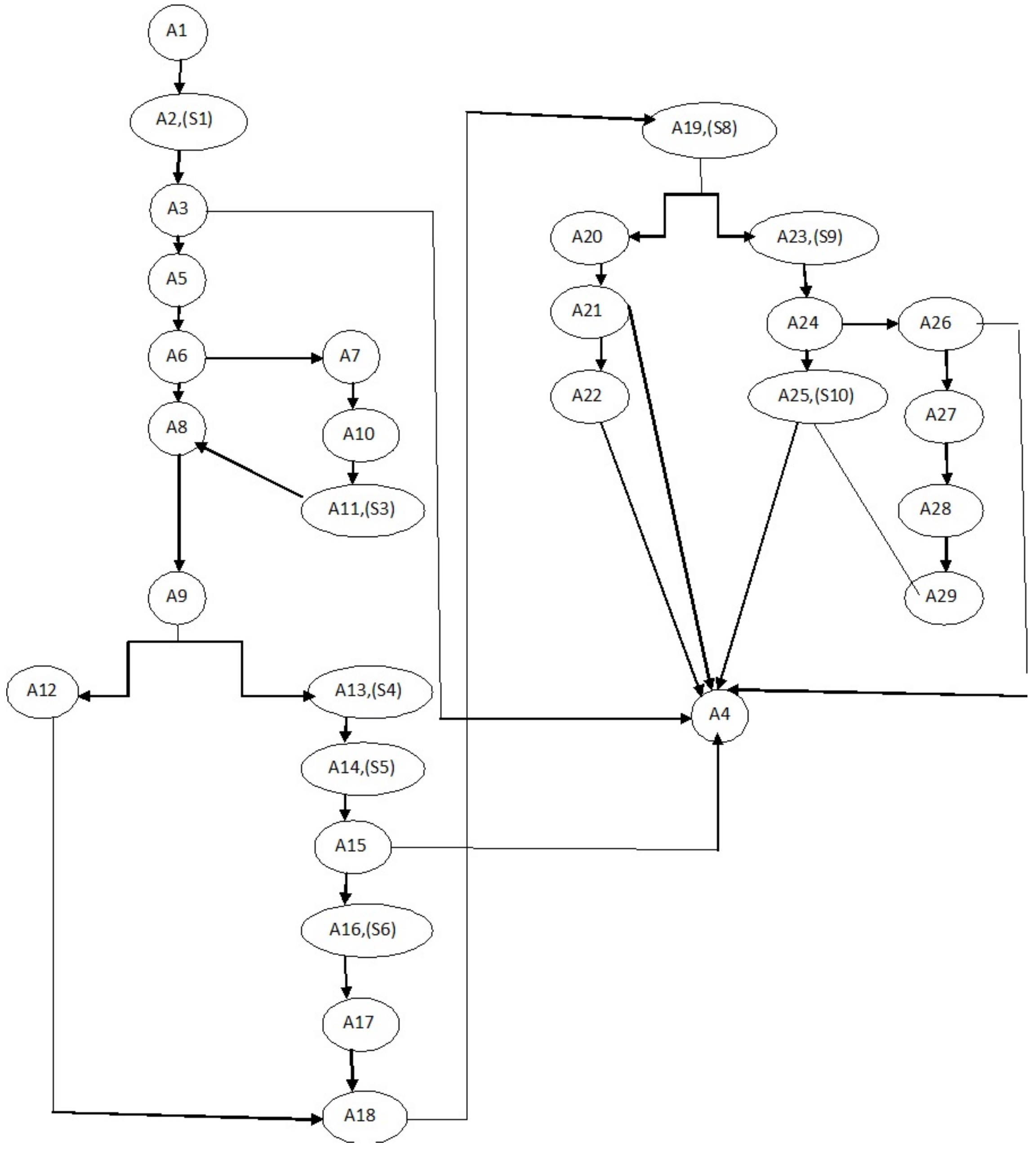


Fig 4.1.3 Activity Sequence Graph of BBAS

4.1.3 GENERATE TEST CASES

The ASG is traversed by using a graph traversal algorithm to find out the linearly independent paths. These independent paths serve as the test scenarios. ASG is traversed from source to destination to generate linearly independent paths and that paths are considered as test cases / scenarios. We have used the terms ‘‘test case’’ and ‘‘test scenario”. Now, the activity sequence graph of BBAS, SMMS, and LMS is used for test scenarios generation by applying graph traversal algorithm. The graph generates 34, 32, 20 test scenarios and these test scenarios represent single paths in the activity sequence graph. The test cases of BBAS are shown in Table 4.1.1

S.NO	Test case id	Independent path
1	TC1	A1-A2(S1)-A3-A4
2	TC2	A1-A2(S1)-A3-A5-A6-A7-A9-A12-A16(S6)-A17-A19(S8)-A20-A21-A22-A4
3	TC3	A1-A2(S1)-A3-A5-A6-A7-A9-A12-A16(S6)-A17-A19(S8)-A23(S9)-A24-A25(S10)-A4
4	TC4	A1-A2(S1)-A3-A5-A6-A7--A9-A12-A16(S6)-A17-A19(S8)-A20-A23(S9)-A24-A26-A27-A28-A29-A25(S10)-A4
5	TC5	A1-A2(S1)-A3-A5-A6-A7-A9-A13(S4)-A14(S5)-A15-A16(S6)-A17-A18-A19(8)-A20-21-A22-A4
6	TC6	A1-A2(S1)-A3-A5-A6-A7-A9-A13(S4)-A14(S5)-A15-A16(S6)-A17-A18-A19(8)-A23(S9)-A24-A25(S10)-A4
7	TC7	A1-A2(S1)-A3-A5-A6-A7-A9-A13(S4)-A14(S5)-A15-A16(S6)-A17-A18-A19(S8)-A23(S9)-A24-A26-A27-A28-A29-A25(S10)-A4
8	TC8	A1-A2(S1)-A3-A5-A6-A7-A9-A12-A16(S6)-A17-A19(S8)-A23(S9)-A24-A26-A4
9	TC9	A1-A2(S1)-A3-A5-A6-A7-A9-A13(S4)-A14(S5)-A15-A16(S6)-A17-A18-A19(8)-A23(S9)-A24-A26-A4
10	TC10	A1-A2(S1)-A3-A5-A6-A7-A9-A12-A16(S6)-A17-A19(S8)-A20-A21-A4
11	TC11	A1-A2(S1)-A3-A5-A6-A7-A9-A13(S4)-A14(S5)-A15-A16(S6)-A17-A18-A19(8)-A20-21-A4
12	TC12	A1-A2(S1)-A3-A5-A6-A7-A10-A11(S3)-A8-A9-A12-A16(S6)-A17-A19(S8)-A20-A21-A22-A4
13	TC13	A1-A2(S1)-A3-A5-A6-A7-A10-A11(S3)-A8-A9-A12-A16(S6)-A17-A19(S8)-A20-A21-A4
14	TC14	A1-A2(S1)-A3-A5-A6-A7-A10-A11(S3)-A8-A9-A13(S4)-A14(S5)-A15-A16(S6)-A17-A18-A19(S8)-A23(S9)-A24-A25(S10)-A4

15	TC15	A1-A2(S1)-A3-A5-A6-A7-A10-A11(S3)-A8-A9-A13(S4)-A14(S5)-A15-A16(S6)-A17-A18-A19(S8)-A23(S9)-A24- A26-A27-A28-A29-A25(S10)-A4
16	TC16	A1-A2(S1)-A3-A5-A6-A7-A10-A11(S3)-A8-A9-A13(S4)-A14(S5)-A15-A16(S6)-A17-A18-A19(S8)-A23(S9)-A24-A26-A4
17	TC17	A1-A2(S1)-A3-A5-A6-A7-A10-A11(S3)-A8-A9-A12-A16(S6)-A17-A19(S8)-A23(S9)-A24-A25(S10)-A4
18	TC18	A1-A2(S1)-A3-A5-A6-A7-A10-A11(S3)-A8-A9-A12-A16(S6)-A17-A19(S8)-A23(S9)-A24-A26-A27-A28A29-A25(S10)-A4
19	TC19	A1-A2(S1)-A3-A5-A6-A7-A10-A11(S3)-A8-A9-A12-A16(S6)-A17-A19(S8)-A23(S9)-A24-A26-A4
20	TC20	A1-A2(S1)-A3-A5-A6-A7-A10-A11(S3)-A8-A9-A13(S4)-A14(S5)-A15-A16(S6)-A17-A18-A19(8)-A20-A21-A22-A4
21	TC21	A1-A2(S1)-A3-A5-A6-A7-A10-A11(S3)-A8-A9-A13(S4)-A14(S5)-A15-A16(S6)-A17-A18-A19(S8)-A20-A21-A4
22	TC22	A1-A2(S1)-A3-A5-A6-A8-A9-A12-A16(S6)-A17-A19(S8)-A20-A21-A22-A4
23	TC23	A1-A2(S1)-A3-A5-A6-A8-A9-A12-A16(S6)-A17-A19(S8)-A20-A21-A4
24	TC24	A1-A2(S1)-A3-A5-A6-A8-A9-A12-A16(S6)-A17-A19(S8)-A23(S9)-A24-A25(S10)-A4
25	TC25	A1-A2(S1)-A3-A5-A6-A8-A9-A12-A16(S6)-A17-A19(S8)-A23(S9)-A24-A26-A27-A28-A29-A25(S10)-A4
26	TC26	A1-A2(S1)-A3-A5-A6-A8-A9-A12-A16(S6)-A17-A19(S8)-A23(S9)-A24-A26-A4
27	TC27	A1-A2(S1)-A3-A5-A6-A8-A9-A13(S4)-A14(S5)-A15-A16(S6)-A17-A18-A19(S8)-A20-A21-A22-A4
28	TC28	A1-A2(S1)-A3-A5-A6-A8-A9-A13(S4)-A14(S5)-A15-A16(S6)-A17-A18-A19(S8)-A20-A21-A4
29	TC29	A1-A2(S1)-A3-A5-A6-A8-A9-A13(S4)-A14(S5)-A15-A16(S6)-A17-A18-A19(S8)-A23(S9)-A24-A25(S10)-A4
30	TC30	A1-A2(S1)-A3-A5-A6-A8-A9-A13(S4)-A14(S5)-A15-A16(S6)-A17-A18-A19(S8)-A23(S9)-A24-A26-A27-A28-A29-A25(S10)-A4
31	TC31	A1-A2(S1)-A3-A5-A6-A8-A9-A13(S4)-A14(S5)-A15-A16(S6)-A17-A18-A19(S8)- A23(A9)-A24-A26-A4
32	TC32	A1-A2(S1)-A3-A5-A6-A8-A9-A13(S4)-A14(S5)-A15-A4
33	TC33	A1-A2(S1)-A3-A5-A6-A7-A9-A13(S4)-A14(S5)-A15-A4
34	TC34	A1-A2(S1)-A3-A5-A6-A7-A10-A11(S3)-A8-A9-A13(S4)-A14(S5)-A15-A4

Table 4.1.1 Test Cases for BBAS

4.1.4 FORWARD SLICING ALGORITHM

Identify all the modified nodes for the respective changes (requirement modifications) and affected nodes with respect to all modified nodes using the proposed forward slicing algorithm given in Algorithm 1 and store the information in the project repository. The collected data from Activity Sequence Graph (ASG) is termed as changes.

Algorithm 1 Forward slicing algorithm

```
Input: Activity Sequence Graph (AG) and total no. of modified nodes (n)
Output: Set of affected nodes for each modified node
1: for $p$ = 0 to $n$ do
2: scanf ("%d", A[p]); // A[p] stores the address of modified nodes and n is the total number of modified nodes.
3: end for
4: for $i$ = 0 to $n$ do
5: tnode = A[i]; // tnode = target node
6: Traverse the graph from tnode using graph traversal algorithm to find the depended nodes ($N1, N2, N3, ....$);
7: AN←− $\{N1, N2, N3, .......Nm\}$; // AN is the set of affected nodes for the modified nodes present in A[i] and m is the total number of affected nodes.
8: printf ("%d", AN);
9: end for
```

Algorithm 1 takes ASG of BBAS as the initial input. It also takes all the modified nodes as input and stores them in an array. Then, the dependent nodes are determined by traversing the graph using the graph traversal algorithm and are stored in an array and treated as the set of affected nodes.

4.1.5 FIND AFFECTED NODES USING FORWARD SLICING ALGORITHM

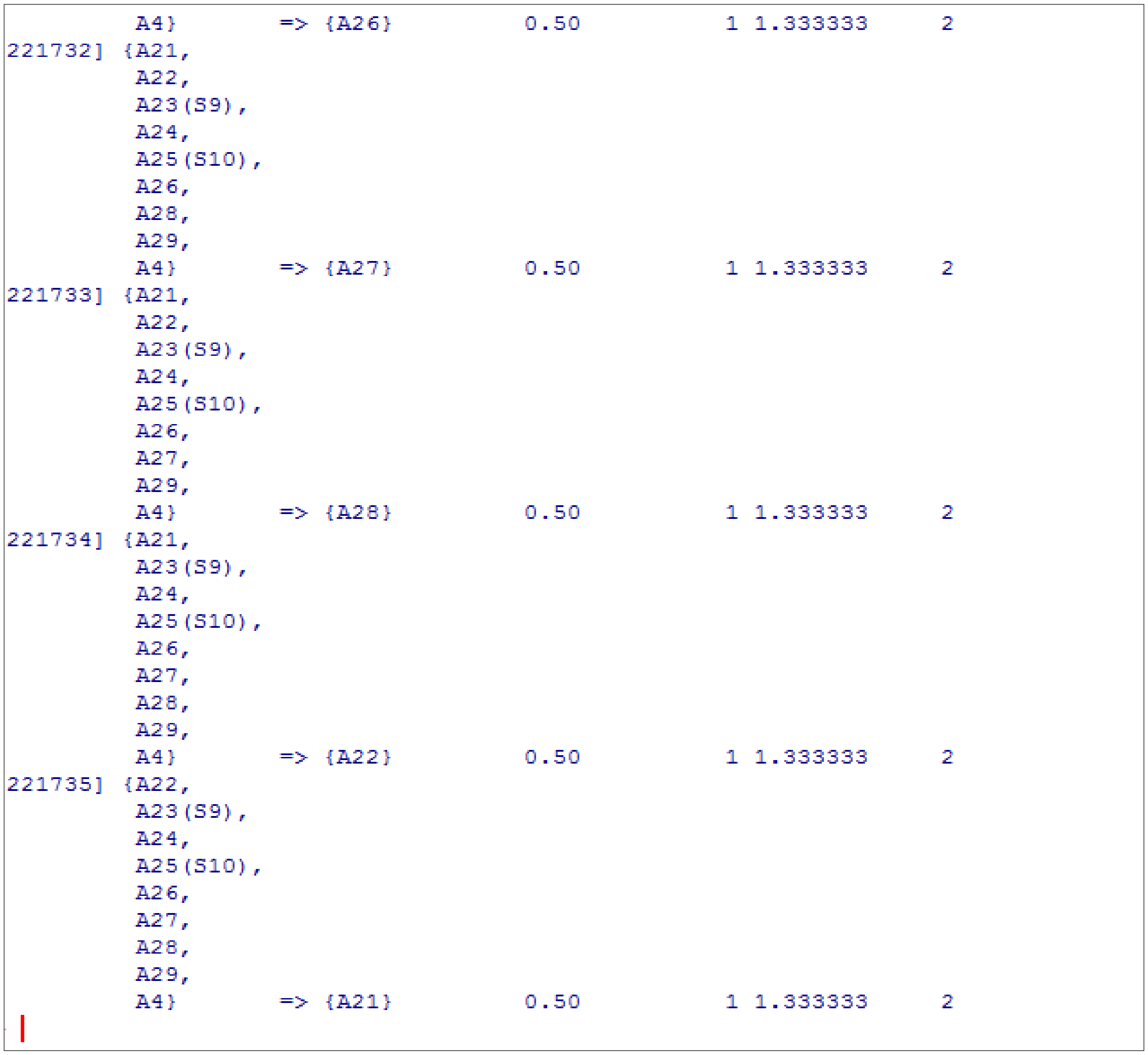
Modified nodes are used to find out a common pattern of affected nodes revealing more number of faults. Then, test case prioritization is done by using the patterns of affected nodes generated using ARM. In this phase, all the modified nodes and the corresponding affected nodes are identified. These nodes are found by using the proposed forward slicing algorithm (given in Algorithm 1). The modified nodes and corresponding affected nodes with respect to different types of changes are given in Table 4.1.2.Then the modified information is saved in the project repository. These sets of nodes are called item set.

Changes	**Modified nodes**	**Affected nodes**
C1	A5	A6,A7,A8,A9,A10,A11(S3),A12,A13(S4),A14(S5),A15, A16(S6),A17,A19(S8),A20,A21,A22,A23(S9),A24,A25(S10),A26,A27,A28,A29,A4
C2	A15	A15,A16(S6),A17,A18,A19(S8),A20,A21,A22,A23(S9), A24,A25(S10),A26,A27,A28,A29,A4
C3	A20	A20,A21,A22,A4
C4	A23	A23(S9),A24,A25(S10),A26,A27,A28,A29,A4

Table 4.1.2 Modified and Affected Nodes of BBAS

4.1.6 GENERATE FPAN USING ARM

For frequent pattern generation, all the modified nodes and associated affected nodes are imported as input and we get a pattern consisting of nodes, as the result. Then, these nodes are imported to ARM technique. To generate a frequent pattern using ARM, two parameters are required i.e. support value and confidence value. First, the SV of individual items (i.e. nodes) and combination of items are determined. Then, the final value is compared with the minimum or threshold support value and the derived item set is known as frequent item set. Similarly, the CV of each frequent item set is determined and also compared with threshold CV. Then, the generated final pattern is known as Frequent Pattern of Affected Node (FPAN). Here, we have implemented the item sets in Ri386 3.5.2 to produce a frequent pattern by taking minimum support value as 40% and confidence value as 80%. The FPAN from the implementation is shown in Fig 4.1.4.



```
         A4}        => {A26}         0.50          1 1.333333     2
221732] {A21,
         A22,
         A23(S9),
         A24,
         A25(S10),
         A26,
         A28,
         A29,
         A4}        => {A27}         0.50          1 1.333333     2
221733] {A21,
         A22,
         A23(S9),
         A24,
         A25(S10),
         A26,
         A27,
         A29,
         A4}        => {A28}         0.50          1 1.333333     2
221734] {A21,
         A23(S9),
         A24,
         A25(S10),
         A26,
         A27,
         A28,
         A29,
         A4}        => {A22}         0.50          1 1.333333     2
221735] {A22,
         A23(S9),
         A24,
         A25(S10),
         A26,
         A27,
         A28,
         A29,
         A4}        => {A21}         0.50          1 1.333333     2
```

Fig 4.1.4 FPAN of BBAS

FPAN for Big Bazaar Automation System found from the above rules is given as:

FPAN={A15,A16(S6),A17,A18,A19(S8),A20,A21,A22,A23(S9),A25(S10),A25,A27,A28,A29,A4}

4.1.7 GENERATE BCV OF ALL NODES IN FPAN

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 BCV of all nodes present in the frequent pattern is determined by using the formula given in Eq. 1.

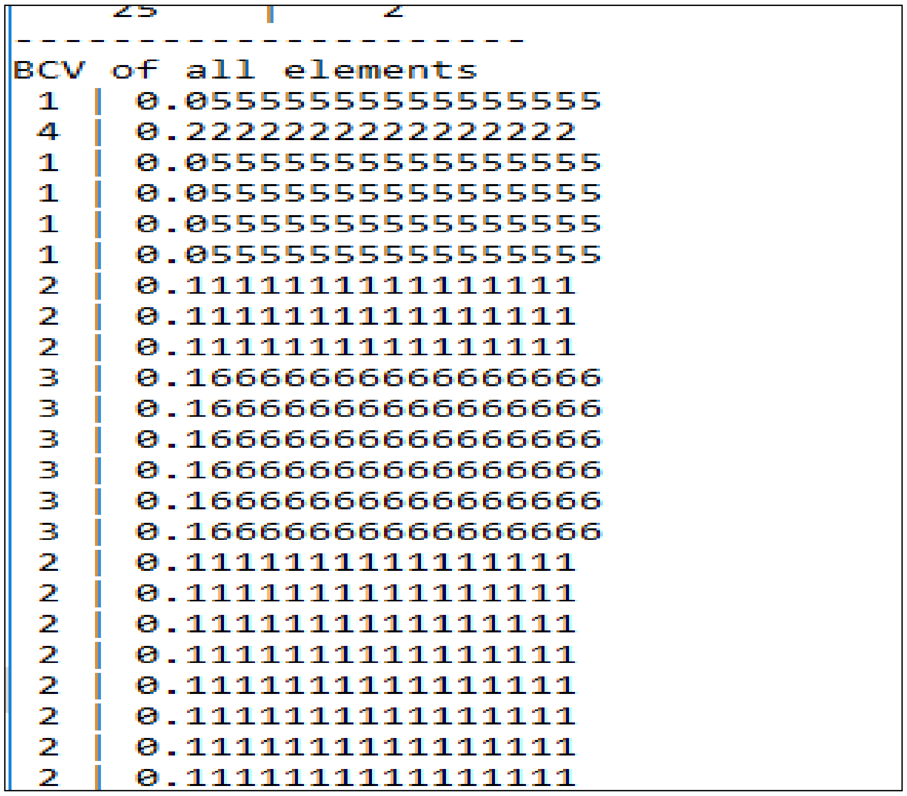
$$BCV = \frac{No\ of\ times\ the\ node\ is\ encountered}{Total\ no\ of\ nodes\ being\ affected} \qquad (1)$$

The BCV calculation of all nodes present in FPAN implemented by java coding and the output is shown in Fig 4.1.5 and Table 4.1.3.

Source code for generating BCV Values

```
public class Bcv {
public static void main(String[] args) {
int [] arr = new int
[]{6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,4,15,16,17
,18,19,20,21,22,23,24,25,26,27,28,29,4,20,21,22,4,23,24,25,26,27,28,29,4};
int [] fr = new int [arr.length];
int visited = -1;
double sum=18.0;
double []  m=new double [fr.length];
for(int i = 0; i < arr.length; i++){
```

```
int count = 1;
for(int j = i+1; j < arr.length; j++){
if(arr[i] == arr[j]){
count++;
fr[j] = visited;
}
}
if(fr[i] != visited)
fr[i] = count;
}
System.out.println("---------------------");
System.out.println(" Element | Frequency");
System.out.println("---------------------");
for(int i = 0; i < fr.length; i++){
if(fr[i] != visited)
 System.out.println("    " + arr[i] + "    |    " + fr[i]);  }
 System.out.println("---------------------");
 System.out.println("BCV of all elements");
for(int i=0;i<fr.length;i++)
{
m[i]=fr[i]/sum;
System.out.println(" "+ fr[i] + " | " + m[i]);
}
}
}
```



```
     25     |     2
---------------------
BCV of all elements
 1  |  0.05555555555555555
 4  |  0.2222222222222222
 1  |  0.05555555555555555
 1  |  0.05555555555555555
 1  |  0.05555555555555555
 1  |  0.05555555555555555
 2  |  0.1111111111111111
 2  |  0.1111111111111111
 2  |  0.1111111111111111
 3  |  0.16666666666666666
 3  |  0.16666666666666666
 3  |  0.16666666666666666
 3  |  0.16666666666666666
 3  |  0.16666666666666666
 3  |  0.16666666666666666
 2  |  0.1111111111111111
 2  |  0.1111111111111111
 2  |  0.1111111111111111
 2  |  0.1111111111111111
 2  |  0.1111111111111111
 2  |  0.1111111111111111
 2  |  0.1111111111111111
 2  |  0.1111111111111111
```

Fig 4.1.5 BCV of BBAS

AFFECTED NODES	NO OF TIMES THE NODE IS ENCOUNTERED	BCV OF NODES
AD4	4	0.2666
AD15	2	0.1333
AD16(S6)	2	0.1333
AD17	2	0.1333
AD18	2	0.1333
AD19(S8)	2	0.1333
AD20	3	0.2
AD21	3	0.2
AD22	3	0.2
AD23(S9)	3	0.2
AD24	3	0.2
AD25(S10)	3	0.2
AD26	3	0.2
AD27	3	0.2
AD28	3	0.2
AD29	3	0.2

Table 4.1.3 BCV for BBAS

4.1.8 GENERATE PRIORITIZED TEST SCENARIOS USING BCTV

In the last step, test cases are prioritized by using BCTV. BCTV of test case is determined by taking the summation of BCV of all nodes executed by the test case.

For example,

TC3=0+0+0+0+0+0+0.133+0.133+0.133+0.33+0.2+0.2+0,2+0.266=1.3998.

The BCTV values of all test cases are given in Table 4.1.4.

The TC containing highest BCTV is considered as highest priority TC and the TC with lowest BCTV is considered as lowest priority TC. The test cases are prioritized based on the priority of BCTV of TCs. Lastly, the prioritized test suite is stored in the project repository for subsequent implementation. TCP is performed by using the BCTV of each test case.

S.NO	TEST SCENARIO IDS	BCTV OF TEST SCENARIOS	PRIORITY OF TEST SCENARIOS
1	TC1	0.2666	8
2	TC2,TC3,TC8,TC12,TC17,TC19,TC22,TC26	1.3998	4
3	TC4,TC18,TC25	2.1998	2
4	TC5,TC6,TC9,TC14,TC16,TC20,TC27,TC31	1.5331	3
5	TC7,TC15,TC29,TC30	2.3331	1
6	TC10,TC13,TC23,TC24	1.1998	6
7	TC32,TC33,TC34	0.3999	7
8	TC11,TC21,TC28	1.3331	5

Table 4.1.4 Prioritized order of test cases in BBAS

If the BCTV of some test cases are equal, then random prioritization technique can be applied to produce the prioritized test cases. So, the prioritized test suite for our case study can be written as follows:

Prioritized order:

{TC7,TC15,TC29,TC30,TC4,TC18,TC25,TC5,TC6,TC9,TC14,TC16,TC20,TC27, TC31,TC2,TC3,TC8,TC12,TC17,TC19,TC22,TC26,TC11,TC21,TC28,TC16,TC1 3,TC23,TC24,TC32,TC33,TC34,TC1}

4.1.9 FAULT DETECTION

Average Percentage of Fault Detection (APFD) metric is available to check the efficiency of test suites. Frequent patterns of nodes revealing more faults which help in identifying the test cases which detects maximum faults, collected from last few releases of the software. If maximum priority is given to retest those test cases first in the next release of software, then we can ensure that the probability of getting defects or faults early will be enhanced and the quality of software will be also improved by the time it is delivered. Hence the software analyst can be able to reduce the estimated budget for the new release of software. The frequent pattern of nodes is discovered by using data mining mechanism.

After performing the TCP, we have to ensure the efficiency and effectiveness of TCs to detect maximum faults or defects. So APFD metric is used to check the efficiency of the prioritized test suite. APFD metric can be expressed as:

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

where T->test suite under evaluation, m->number of faults contained in the program, n -> total number of test cases, TFi -> position of first test case in T that Exposes fault i. Due to modification, the current version of the system detects Different types of faults. (Given in Table 4.1.5)

	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	F15	F16	F17	F18	F19	F20
TC1	*																			
TC2						*											*			
TC3				*									*							
TC4	*	*										*					*	*		
TC5																*				*
TC6								*		*							*			
TC7					*	*														
TC8				*	*	*				*	*									
TC9	*						*	*			*									
TC10							*			*										
TC11	*		*			*						*								
TC12								*		*										
TC13							*			*										
TC14			*		*							*								
TC15	*					*							*	*						
TC16	*							*										*		
TC17		*									*				*					
TC18								*							*	*				
TC19				*					*								*			
TC20				*												*				
TC21	*					*	*						*							
TC22		*									*									*
TC23				*	*				*											
TC24			*				*									*				
TC25	*					*	*						*							
TC26		*			*					*							*			
TC27					*				*											
TC28						*						*			*					
TC29			*					*										*	*	
TC30		*							*											
TC31					*					*			*		*					
TC32					*				*											*
TC33		*			*			*												
TC34		*								*										

Table 4.1.5 Total number of faults detected by each test case in BBAS

For the Non Prioritized test suite,

$$APFD = 1 - \frac{1+4+11+3+7+2+9+6+19+6+8+4+3+15+17+5+2+4+29+5}{20 \times 34} + \frac{1}{2 \times 34}$$

= 1-0.2352+0.0147=0.7795

For the Prioritized test suite,

$$APFD = 1 - \frac{2+4+3+13+1+1+7+3+4+9+10+5+2+2+6+6+5+5+3+8}{20 \times 34} + \frac{1}{2 \times 34}$$

$=1-0.1455+0.0147=0.8692$

APFD value for the given prioritized test suite is calculated to be 0.8692 and for the non-prioritized test suite is 0.7795. So, it can be inferred that the prioritized test cases for our case study are having better capability to detect more faults early in the software development process than that of the non-prioritized TCs. Figure 4.1.6 shows the APFD values for the prioritized and non-prioritized TCs for our case study using BCV value method.

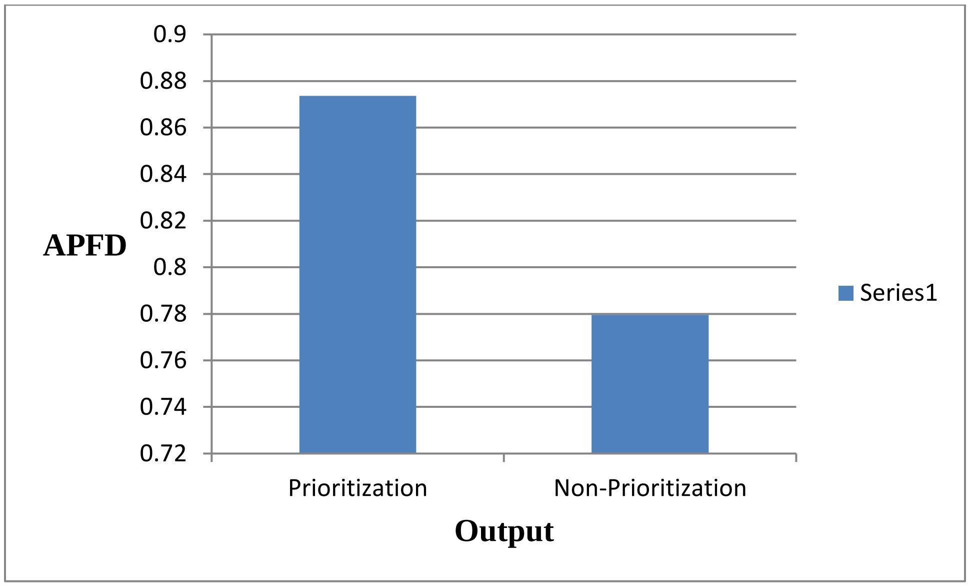


Fig 4.1.6 APFD values for prioritized and non-prioritized test cases Using BCV method of BBAS

4.2 TCP USING WEIGHT METHOD

The first four steps are same as BCV approach. And then

- Calculate weight values of all modified nodes from Table 4.1.3
- Calculate weight values of all edges in ASG.

- Calculate total weight for test cases path (ASG Path).
- Prioritize the test case based on each test cases weight.

4.2.1 CALCULATE NODE WEIGHT

To calculate the weights of each modified node of the graph according to the affecting nodes using forward slice. Then the weights of test paths which are generated from sequence diagram are calculated by adding the weights of associated nodes and edges. According to the weights of corresponding test paths the test cases are prioritized. The obtained results indicate that the proposed technique is effective in prioritizing the test cases by the Average Percentage of Fault Detection (APFD) metric to estimate the performance of our proposed approach.

In our prioritization technique, we apply forward slice to provide the weight to each of the nodes of ASG. This weight represents the number of nodes affected by making modification at that node. Node weight of a node (Ni) in ASG is denoted as the number nodes affected by Ni in the ASG. The calculation of node weight Ni using the given Equation 3. And weight values for all modified nodes present in Table 4.2.1.

$$\text{Weight (Ni)} = \text{No. of affected nodes by (Ni)} \qquad (3)$$

Changes	**Modified nodes**	**Affected nodes**	**Node weight**
C1	A5	A6,A7(S2),A8,A9,A10,A11(S3),A12,A13(S4),A14(S5),A15, A16(S6),A17,A18,A19(S8),A20,A21,A22,A23(S9),A24,A25(S10),A26,A27,A28,A29,A4	25
C2	A15	A15,A16(S6),A17,A18,A19(S8),A20,A21,A22,A23(S9),A24,A25(S10),A26,A27,A28,A29,A4	16
C3	A20	A20,A21,A22,A4	4
C4	A23,(S9)	A23(S9),A24,A25(S10),A26,A27,A28,A29,A4	8

Table 4.2.1 Node Weight in ASG

4.2.2 CALCULATE EDGE WEIGHT

The weight (cost) of edges is assigned using information flow model. This weight represents the strength of message along the path. The cost of each edge is computed using the information flow index of connecting node. So the cost of an edge ei $\in$ E connecting two consecutive nodes Ni and Ni+1of ASG is computed using this formula.

$$\text{Weight (ei)} = \text{OUT (Ni+1)} \tag{4}$$

Where OUT (Ni+1) is the number of outgoing edges of node Ni+1. The weight values of all edges shown in Table 4.2.2.

Edge	Weight
A1-A2	1
A2-A3	2
A3-A4	0
A3-A5	1
A5-A6	2
A6-A7	1
A6-A8	1
A8-A9	2
A7-A10	1
A10-A11	1
A9-A12	1
A9-A13	1
A13-A14	1
A14-A15	2
A15-A16	1
A16-A17	1
A17-A18	2
A18-A19	1
A19-A20	1
A19-A23	2
A20-A21	1
A21-A22	2
A23-A24	1

A24-A25	1
A24-A26	2
A26-A27	1
A27-A28	1
A28-A29	1

Table 4.2.2 Edge Weight in ASG

4.2.3 CALCULATE ACTIVITY SEQUENCE PATH WEIGHT

Activity Sequence Path (Test cases) weight calculated using Equation 5 and is tabulated in Table 4.2.3

$$PW(p_k) = \sum Weight(Ni) + \sum Weight(ei) \qquad (5)$$

Test cases	Node Weight	Edge weight	Total weight
TC1	0	3	3
TC2	29	19	48
TC3	33	20	53
TC4	33	24	57
TC5	46	22	68
TC6	50	23	73
TC7	50	27	77
TC8	33	20	53
TC9	50	23	73
TC10	29	18	47
TC11	46	21	67
TC12	29	22	51
TC13	29	21	50
TC14	50	25	75
TC15	50	30	80
TC16	50	26	76
TC17	33	23	56
TC18	33	26	59
TC19	33	23	56
TC20	46	25	71
TC21	46	24	70
TC22	29	18	47
TC23	29	17	46

TC24	33	18	51
TC25	33	23	56
TC26	33	20	53
TC27	46	21	67
TC28	46	20	66
TC29	50	21	71
TC30	50	26	76
TC31	50	22	72
TC32	42	12	54
TC33	42	13	55
TC34	42	17	59

Table 4.2.3 Total Weight in Test Cases

4.2.4 PRIORITIZED ORDER OF TEST CASES

Then, the prioritizations of test cases are to be made on the order of activity sequence path weight value.

Total weight	Test cases	Prioritization of test cases
3	TC1	23
48	TC2	20
53	TC3,TC8,TC26	17
57	TC4	13
68	TC5	9
73	TC6,TC9	5
77	TC7	2
47	TC10,TC22	21
67	TC11,TC27	10
51	TC12,TC24	18
50	TC13	19
75	TC14	4
80	TC15	1
76	TC16,TC30	3
56	TC17,TC19,TC25	14
59	TC18,TC34	12
71	TC20,TC29	7
70	TC21	8
46	TC23	22
66	TC28	11

72	TC31	6
54	TC32	16
55	TC33	15

Table 4.2.4 Prioritized order of test cases

Prioritized order:

TC15,TC7,TC16,TC30,TC14,TC6,TC9,TC31,TC20,TC29,TC21,TC5,TC11,TC27,TC28,TC18,TC34,TC4,TC17,TC19,TC25,TC33,TC32,TC3,TC8,TC26,TC12,TC24,TC13,T2,TC10,TC22,TC23,TC11

4.2.5 FAULT DETECTION

We have used APFD metric to show the increased rate of fault detection of a test suite quantitatively. It measures the percentage of faults detected using weighted average. The rate of fault detection is faster if the value of APFD is higher. In order to achieve better performance in software testing, we presented a TCP (Test Case Prioritization) technique to schedule the order of test cases generated from UML activity and sequence diagrams. Teat case prioritization is used, to increasing higher rate of detecting faults, and faster rate in increasing the confidence in reliability of the system.

APFD for prioritized test cases

$$APFD = 1 - \frac{1+4+5+9+2+1+7+3+4+6+7+5+1+1+8+9+6+3+8+6}{20 \times 34} + \frac{1}{2 \times 34}$$

=1-0.1411+0.0147=0.8736

APFD value for the given prioritized test suite is calculated to be 0.8736 and for the non-prioritized test suite is same as BCV method.

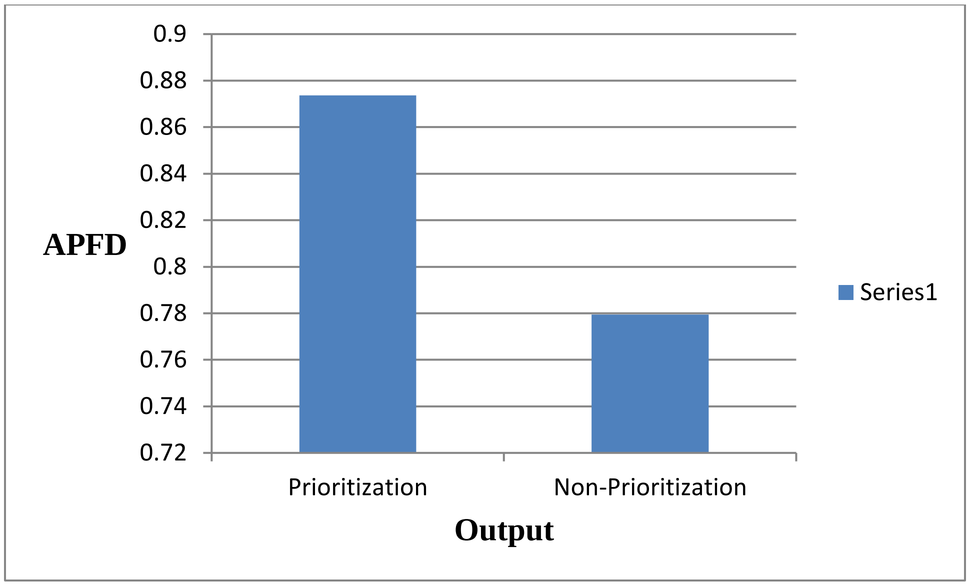


Fig 4.2.1 APFD values for prioritized and non-prioritized test cases Using Weight method in BBAS

CHAPTER 5
DISCUSSION ON EXPERIMENTAL RESULTS

5.1 EFFECTIVENESS USING APFD METRIC

To further validate our approach, we have considered two other case studies i.e. Shopping Mall Management System (SMMS), Library Management System (LMS). The respective APFD values of prioritized and non-prioritized TCs calculated by BCV and weight approach for these case studies are shown in Table 5.1, 5.2 and Fig 5.1, 5.2. It is observed that the APFD values of prioritized TCs are greater than the APFD values of non-prioritized TCs.

Researchers found many kinds of challenges in code based testing like difficulties in deriving the dependencies among different functions and among the classes, extraction of lines of code from components etc. To eliminate these problems, researchers came up with the proposal of test case prioritization in the context of MBT (Sarma and Mall 2007; Swain and Mohapatra 2010; Muthusamy and Seetharaman 2014; Indumathi and Selvamani 2015). The dependencies among the different functions or classes are very much visible in case of model based testing. They help in test case prioritization to identify the most appropriate test cases which reveal maximum number of faults. But, they have not considered the severity of faults as a factor for prioritization. Also, the faults which were detected are not saved in any database for future reference. So, the behavior of the previous versions of the system is not taken into account. After that, researchers came up with an idea of maintaining a historical data store and mining the data store to get the frequent patterns and more appropriate test cases which reveal more number of faults. The APFD value is also increased in this case. But, all the MBTP techniques are using single UML diagram for system modeling. So, the type of faults detected by any test case is limited. We have compared our two approaches with existing approach considering the same case studies. In our proposed approach provides higher quality results than that of others.

Sl. No	System name	Number of test cases	Prioritized APFD value	Non-prioritized APFD value
1	Big Bazaar Automation System	34	0.8692	0.7795
2	Shopping mall Management System	32	0.8328	0.7362
3	Library Management System	20	0.7375	0.6325

Table 5.1 APFD values for different case studies using BCV approach

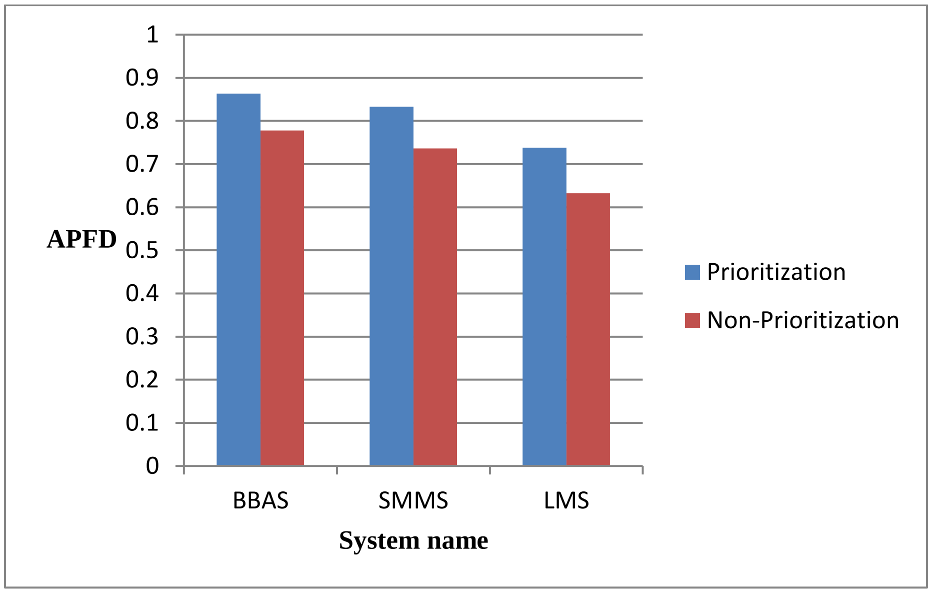


Fig 5.1 APFD values for different case studies using BCV approach

Sl. No	System name	Number of test cases	Prioritized APFD value	Non-prioritized APFD value
1	Big Bazaar Automation System	34	0.8736	0.7795
2	Shopping mall Management System	32	0.85	0.7362
3	Library Management System	20	0.7625	0.6325

Table 5.2 APFD values for different case studies using Weight approach

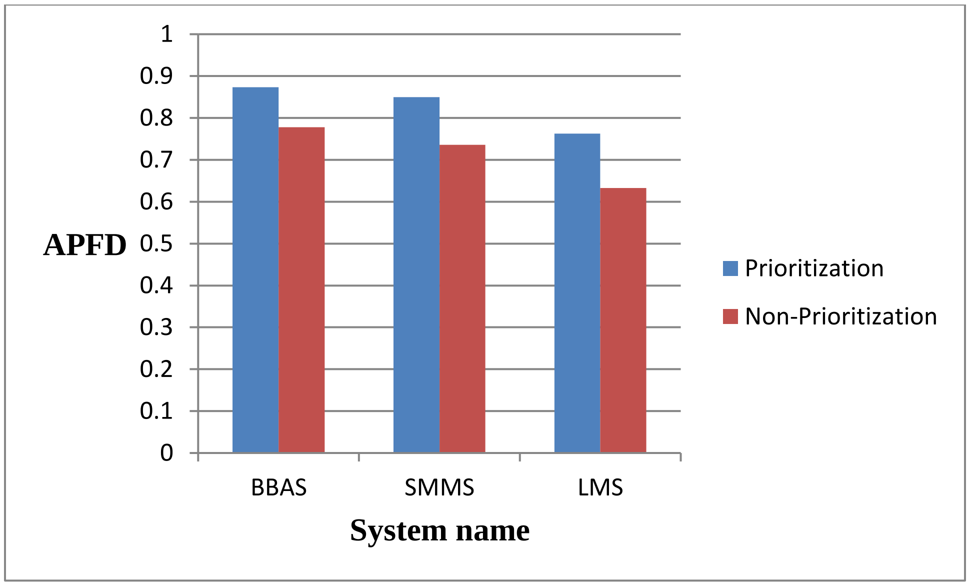


Fig 5.2 APFD values for different case studies using Weight approach

5.2 COMPARATIVE STUDY

The comparison of results of our proposed approach with existing approach using APFD for different selected software is given in Table 5.3 from Table 5.3; it is observed that our bi-graph approach helps to increase the percentage of fault detection rate as compared to the existing approach. Comparison of existing approach vs. proposed approach is shown in Figure 5.3.

DIFFERENT SYSTEMS	BBAS	SMMS	LMS
Existing work	0.3808	0.7786	0.4635
Proposed work using BCTV	0.8692	0.8328	0.7375
Proposed work using Weight value	0.8736	0.85	0.7625

Table 5.3 APFD Values for Existing Vs Proposed work

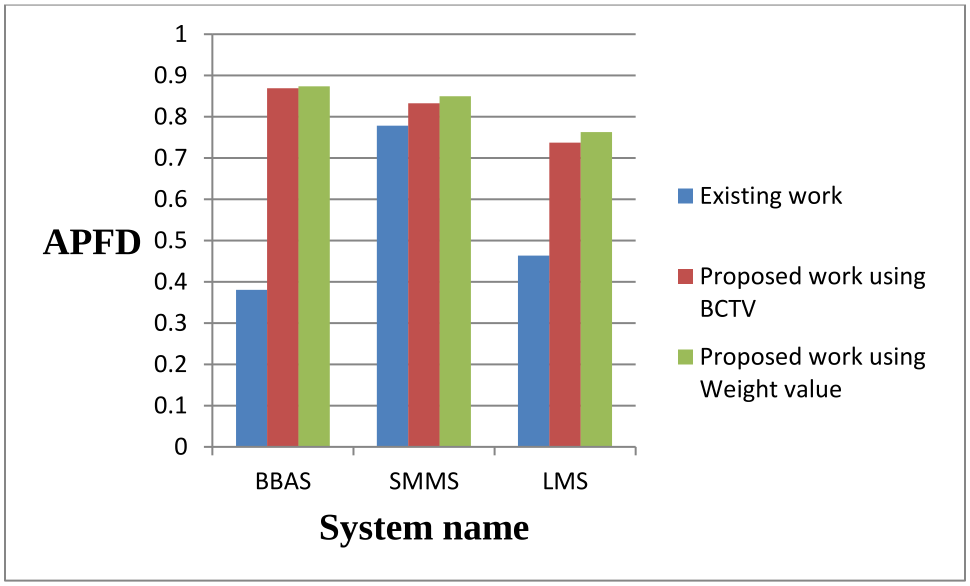


Fig 5.3 Existing vs. Proposed Work –APFD values

CHAPTER 6
CONCLUSION

Test case Prioritization method, to generate the test cases automatically from UML activity and sequence diagrams. The UML models into an intermediate representation called Activity Sequence Graph (ASG). Frequent patterns generated using association rule mining helped effectively in prioritization. Using BCV and BCTV method to prioritizing the test cases to increasing the rate of fault detection. This approach gives a better result with respect to the average percentage of fault detection. The proposed framework for TCP is also applied on several case studies and found to be very useful in early fault detection. Another method is weight method, to prioritizing the test cases to increasing the rate of fault detection. The result of our proposed approach is compared with the result of existing approach to increasing the rate of fault detection.

FUTURE WORK

In future, we will consider the non-functional aspects while prioritizing the test cases. Similarly, the other data mining techniques such as apriori algorithm, FP growth etc. can be used for TCP. The results obtained from our approach are compared with the approaches of some other researches. It is observed that, our approach performs better then the randomized approach & some existing approaches. In future work, we would like to optimize the test cases using some soft computing techniques like deep learning and artificial neural network etc.

REFERENCES

1. IEEE Standard Glossary of Software Engineering Terminology. www.ieeexplore.ieee.org/ IEEE Standard Glossary of Software Engineering Terminology(1990).
2. Aggrawal KK, Singh Y, Kaur A (2004) Code coverage based technique for prioritizing test cases for regression testing. ACM SIGSOFT Softw Eng Notes 29(5):1–4.
3. Sarma M, Mall R (2007) Automatic test case generation from UML models. In: Proceedings of 10th international conference on information Technology, pp 196-201.
4. Mathur AP (2008) Foundations of Software Testing, 1st edn. Addison-Wesley professional, Boston.
5. Korel B, Koutsogiannakis G (2009) Experimental comparison of code-based and model-based test prioritization. In: Proceedings of IEEE international conference on software testing verification and validation workshops, pp 77–84.
6. Han J, Kamber M (2010) Data mining: concepts and techniques. Morgan Kaufmann Publishers, 500 Sansome Street, Suite 400, San Francisco, CA 94111, 2nd edition.
7. Askarunisa A, Shanmugariya L, Ramaraj N (2010) Cost and coverage metrics for measuring the effectiveness of test case prioritization techniques. INFOCOMP J Comput Sci 9(1):43–52.
8. Coremen TH, Leiserson CE , Revest RL ,Stein C (2010) Introduction to algorithms, 2nd edn. PHI Learning Private Limited , New Delhi.
9. Swain SK , Mohapatra DP (2010) Test case generation from behavioural UML Models. Int J Computer Appl 6(8): 5-11.

10. Khandai S, Acharya AA, Mohapatra DP (2011) Prioritizing test cases using business test criticality value. Int J Adv Comput Sci Appl 3(5):103-110.
11. Pandey AK, Shrivastava V (2011) Early fault detection model using integrated and cost-effective test case prioritization. Int J Syst Assur Eng Manag 2(1):41–47.
12. Garg D, Datta A, French T (2012) New Test case prioritization Strategies for regression testing of web applications. Int J Syst Assur Eng Mnang 3(4): 300-309.
13. Han X, Zeng H, Gao H (2012) A heuristic model-based test prioritization method for regression testing. In: Proceedings of international symposium on computer, consumer and control, IEEE, pp 886–889.
14. Huang Y-C, Peng K-i, Huang C-Y (2012) A history-based cost cognizant test case prioritization in regression testing. J Syst Softw 85:626–637.
15. Khalilian A, Azgomi MA, Fazlalizadeh Y (2012) A improved method for test case prioritization by incorporating historical test data. Sci Comput Program 78:93–116.
16. Shahid M, Ibrahim S (2014) A new code based test case prioritization technique. Int J Softw Eng Appl 8(6):31–38.
17. Muthusamy T, Seetharaman K (2014) A new effective test case prioritization for regression testing based on prioritization algorithm. Int Appl Inf Syst (IJAIS) 6(7):21–26.
18. Mall R (2014) Fundamental of software engineering, 4th edn. PHI Learning Private Limited, New Delhi.
19. Acharya AA, Mahali P, Mohapatra DP (2015) Model based test case prioritization using association rule mining. Comput Intell Data Min 3:429–440.

20. Indumathi CP, Selvamani K (2015) Test case prioritization using open dependency structure algorithm. In: Proceedings of international conference on intelligent computing, communication and convergence (ICCC-2015), Procedia Computer Science, vol 48. Elsevier, pp 250–255.
21. Tyagi M, Malhotra S (2015) An approach for test case prioritization based on three factors. Int J Inf Technol Comput Sci 4:79–86.
22. A Besz´edes, "Global dynamic slicing for the C language," Acta Polytechnica Hungarica, vol. 12, no. 1, pp. 117–136, 2015.
23. Wang X, Zeng H (2016) History-based dynamic test case prioritization for requirement properties in regression testing. In: Proceedings of international workshop on continuous software evolution and delivery, Austin, pp 41–47.
24. Rava M, Wan-Kadir WMN (2016) A review on prioritization techniques in regression testing. Int J Softw Eng Appl 10(1):221–232.
25. Chauhan N (2016) Software testing principles : practices, 2nd edn.Oxford University Press, New Delhi.
26. S. Panda, D. Munjal, D. P. Mohapatra, ”A Slice-Based Change Impact Analysis for Regression Test Case Prioritization of Object-Oriented Programs”, Advances in Software Engineering, Volume 2016, pp.1-20, 2016.
27. Solanki S (2017) A review an advance approach for test case prioritization for regression testing. Int J Emerg Trends Technol Comput Sci (IJETTCS) 6(1):62–65.
28. S.S.Basa, S.Swain, D.P.Mohapatra,”UML Activity Diagram-Based Test Case Generation”, Journal of Emerging Technologies and Innovative Research (JETIR), Volume 5, Issue 8, 2018.
29. J. Chen , L. Zhu , T. Y. Chen , D. Towey, Fei-Ching Kuo , R. Huang , Y. Guo, "Test case prioritization for object-oriented software: An adaptive

random sequence approach based on clustering*”,* The Journal of Systems and Software, Vol. 135, pp. 107–125, 2018.

30. M. Khatibsyarbini, M.A. Isa, Dayang N.A, Jawawi, R. Tumeng, “Test case prioritization approaches in regression testing: A systematic literature review*”* Information and Software Technology, Vol.93, pp. 74-93, January 2018.

PUBLICATION DETAILS

[1] S. Monisha & Dr. C .P Indumathi and K. Ponmalar, “Bi-Graph Approach for Test Case Prioritization”, published the paper in the proceedings of “UGC-SAP National Conference on Pattern Recognition, Informatics and Medical Engineering (PRIME-2019)”, paper ID: PRIME 19014 p.no:8, Periyar University ,Salem
21st -22nd March 2019.