Int J Syst Assur Eng Manag (October 2018) 9(5):1063–1079
https://doi.org/10.1007/s13198-018-0736-7

ORIGINAL ARTICLE

Model based test case prioritization using UML behavioural diagrams and association rule mining

Prateeva Mahali[1] · Durga Prasad Mohapatra[1]

Received: 1 June 2017 / Revised: 12 January 2018 / Published online: 21 July 2018

Abstract In software development life cycle, maximum effort is spent on the maintenance phase. This is due to the retesting carried out in this phase to ensure that any moderation made to the system under test (SUT) does not hamper the unchanged components of the SUT. This retesting is a part of regression testing which is performed in the maintenance phase. But in the retesting approach, all the old test cases are re-executed which leads to increase in cost and time of testing. So, test case prioritization technique is widely used to overcome this problem i.e. to keep the testing cost and time down. Test case prioritization techniques schedule the test cases for regression testing in an order that improves rate of fault detection, coverage percentage etc. To improve the fault detection rate, we propose an approach for prioritizing the test cases by using multiple modified functions and association rule mining. Since, we are doing model based testing, UML (Unified Modelling Language) behavioural diagrams such as activity diagram and sequence diagram are used to model the system. An activity sequence graph (ASG) is generated taking into account the combined features of activity diagram and sequence diagram. Then, test scenarios are generated by traversing the graph. The affected nodes and corresponding modified nodes are found out using forward slicing algorithm. The details of modified nodes and corresponding affected nodes are stored in a project repository. 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 determining the percentage of fault detection.

✉ Prateeva Mahali
prateevamahali@gmail.com

Durga Prasad Mohapatra
durga@nitrkl.ac.in

1 Department of Computer Science & Engineering, National Institute of Technology, Rourkela, India

Keywords Association rule mining · Regression testing · Test case prioritization · UML · Business criticality test value

1 Introduction

Software testing or program testing is a very indispensable phase of Software Development Life Cycle (SDLC). In SDLC, this phase consumes minimum 40% of total development effort and cost (Chauhan 2016; Mall 2014). When the software testers completely test the system, it is delivered to the customer and then feedback for the system are collected. After receiving the feedback, the system undergoes modifications which include enhancement of requirements, modification of existing requirements etc. When the modifications are implemented, the performance of the existing system improves. After making the required changes, the system is again completely retested to ensure that the modification does not affect the existing functionalities of the system. The retesting of the system is called Regression Testing (www.ieeexplore.ieee.org; Chauhan 2016; Mathur 2008). Regression testing is performed in software maintenance phase. Approximately 60% of organizational software effort is spent in maintenance phase for retesting of the system (Chauhan 2016; Mall 2014; Mathur 2008). For complex and critical systems, the estimated effort and cost may be increased.

Therefore, a suitable technique is necessary to scale down the testing time and cost.

To depreciate the regression testing time, cost and effort, various techniques can be used for test case selection, test suite minimization and test case prioritization (Chauhan 2016). The aim of test case selection technique is to choose a group or subset of test cases from the existing test suite to decrease the test suite size. Test suite minimization technique permanently eliminates the redundant test cases from the test suite. But, test case prioritization attempts to reduce both testing time and cost by reordering of test cases so that the most beneficial test case will be executed first without affecting the performance of a system (Rava and Wan-Kadir 2016; Wang and Zeng 2016; Solanki 2017). The reordering of test cases is done by accrediting a priority value to each test case and the priority is decided as per different criteria like early fault detection rate, coverage rate, risk coverage etc. (Aggrawal et al. 2004; Askarunisa et al. 2010; Pandey and Shrivastava 2011; Garg et al. 2012; Tyagi and Malhotra 2015).

We have carried out the survey on different approaches of test case prioritization proposed by various researchers. We have observed that the defects present in test data are also responsible for the failure of the system. In this research work, we discuss a technique for TCP which takes historical data of the system as input and generates a frequent pattern using ARM.

The remaining of this paper is structured as follows: Sect. 2 presents basic concepts and the related work is explained in Sect. 3. Section 4 discusses the detail of proposed approach and Sect. 5 explains illustration of our approach with a case study of Hospital Management System (HMS). Section 6 discusses comparison of our approach with related work and Sect. 7 reports the conclusions of this research and states future works.

2 Basic concepts

Most of the software products are not accepted by the customers because they fail to satisfy the user requirements. After delivery of the product, the manufacturing company collects user feedback to enhance the speciality of the product. Then modification of the product is done as per the collected feedback in the existing product, during the maintenance phase. The modification includes addition of functionalities, changes in the existing functionalities, rechecking the performance of functionalities with the new environment etc. (Mall 2014). The modified product is again retested with the old TCs to assure that the modification is not affecting the existing product. This kind of retesting process is called regression testing. Regression testing includes three techniques i.e. test case selection, test suite minimization and test case prioritization (Chauhan 2016; Mall 2014). Test case selection technique selects a group of test cases and test suite minimization technique permanently eliminates the redundant test cases to decrease the test suite size. The first technique reduces the testing time and the second technique reduces the testing cost. But test case prioritization technique reduces both testing time and cost by prioritizing the test cases to maximize the objective function like early fault detection rate, maximum path coverage and risk coverage etc.

Test case prioritization can be applied in both code based testing and model based testing (Korel and Koutsogiannakis 2009). In CBT, the system source code or lines of code is passed as input to generate the TCs and those test cases are again used for prioritization process. Similarly, in model based testing, the system is used as input to develop TCs and that TCs are passed for prioritization process. In most of the companies, software developers prefer Unified Modelling 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. Here, TCs are developed from a model dependency graph named ASG which is developed by combining both the diagrams. Then, the modified nodes and associated affected nodes are assembled using an algorithm. The frequent patterns of affected nodes are produced using a data mining technique called association rule mining (ARM). Subsequently, BCV of all the nodes available in the frequent pattern and BCTV of all the generated TCs are determined. Finally, prioritization of test case is performed with the help of BCTV.

Association rule is a popular method for determining exceptional relations, associations and frequent patterns between data in large database. The aim of this rule is to determine the consistency between the products in large scale database. In an association rule mining, a rule is defined as an implication form $M \Rightarrow N$ where $M, N \subseteq S$ and $M \cap N = \phi$, S is the itemset (Han and Kamber 2010). Here, it is required to satisfy two parameters simultaneously: user-specified minimum *support value* and user-specified minimum *confidence value*.

– In association rule, the *support* parameter is written in the form of $M \Rightarrow N$. It gives an indication of how frequently the items appear in the database. Support is represented in the form of probability notation P($M \cup N$), which defines the proportion of transactions in the data set which contain both the itemsets i.e. the union of itemsets M and N (Han and Kamber 2010). Mathematically, $support(M \Rightarrow N) = P(M \cup N)$
– Another measuring parameter for association rule is *confidence*. Confidence can be interpreted as an

estimate of the conditional probability P($N \mid M$) i.e. the proportion of transactions that contains M which also contains N (Han and Kamber 2010).
Mathematically,
$confidence(M \Rightarrow N) = P(N|M) = \frac{support(M \cup N)}{support(M)}$

There are two separate steps to generate association rule.

Step-1 Frequent itemsets in a database are determined by applying minimum support value.

Step-2 The resultant frequent itemsets and minimum or threshold confidence value are used to form the rules.

Example - For a given database transaction buy(Apple,"Orange")⇒buy(Apple,"Banana") with support = 40% and confidence = 80% leads to the conclusion that Orange and Banana are purchased simultaneously by 40% of all transactions and 80% of all consumers.

Along with the association rule mining technique, BCV of all the functions is calculated. It is defined as the extent of function contribution towards the business process of the system under test for achieving better result. The BCV of the function is calculated as follows:

- Identify the functions which are distressed or altered due to the modification done in the project.
- Determine average interaction of various functions within the system.

The abbreviations of the terms used in the paper are given in Table 1.

Table 1 List of abbreviations

Sl no.	Abbreviation	Description
1	AD	Activity diagram
2	SD	Sequence diagram
3	ASG	Activity sequence graph
4	TC	Test case/scenario
5	TCP	Test case prioritization
6	ARM	Association rule mining
7	GD	Graph data
8	OD	Observation data
9	HMS	Hospital Management System
10	MBT	Model based testing
11	CBT	Code based testing
12	FP	Frequent pattern
13	FPAN	Frequent pattern affected node
14	BCV	Business criticality value
15	BCTV	Business criticality test value
16	UML	Unified Modelling Language
17	APFD	Average percentage of fault detection
18	MBTP	Model based test case prioritization
19	GA	Genetic algorithm
20	SV	Support value
21	CV	Confidence value

3 Related work

Here, we discuss few relevant approaches to this proposed approach. All the approaches are confined to test case generation and prioritization in model-based testing.

Han et al. (2012) discussed a new heuristic approach for MBTP in regression testing. The authors have developed this prioritization technique to detect faults within a short time period. Then, test cases are prioritized base on fault detection rate. They have also used APFD metric to compare different TCP methods with this approach. In regression testing, they have taken Extended Finite State Machine (EFSM) model and they proposed two heuristic models for prioritizing the test case.

Khandai et al. (2011) presented a technique for prioritization of test cases in MBT. Here, the authors have maintained a registry which contain the complete information of the undergoing project. Then, the latest configuration of the project is compared with the older configuration and a value is assigned to each affected function, called business criticality value. At last, test case prioritization was done by calculating BCTV of each test case. The disadvantage of this approach is that only functional requirements of the project was taken up for criticality value calculation. But, some non-functional requirements are also present in every project and those requirements may affect the criticality value of test cases.

Acharya et al. (2015) expressed a framework for TCP in model based testing. Test case prioritization was performed by developing a pattern using ARM. As per the proposed approach, the historical data or information of the system was preserved in a database. The database contained two types of data such as GD and OD. The GD was gathered from the model dependency graph. As a result, the modified nodes and the corresponding affected nodes were collected and stored in that historical database. That affected nodes are used as input to generate a pattern. The pattern was generated by using ARM and used for prioritizing the test cases. The TCP was performed by allocating a priority value to all nodes present in that frequent pattern. But, there is a limitation present in this framework. They have considered only single type of modification of the system. There might have multiple modifications made in different versions of the project.

Muthusamy and Seetharaman (2014) presented a methodology for prioritization of test cases by identifying the critical defects or faults. In this paper, the authors have considered some practical weight factors to prioritize the test cases. The weight factors are customer allotted priority, developer observed code execution complexity, change in requirements, fault impact, completeness and traceability.

Indumathi and Selvamani (2015) discussed some algorithms for test case prioritization using open dependency structure. In this paper, the authors have considered open dependency between the functions in a system as input to the proposed approach. They have found out the exact number of dependants of each function for test case prioritization. Test cases that are more dependent are assigned high priority values and accordingly prioritization of test cases is carried out. The major limitation of this approach is that they have considered a code based approach for test case prioritization where it is very difficult to derive the dependency structure among different functionalities of the system. They have not addressed the closed dependency structure that may exist among the functions. In this paper, first they have found the priority factors for requirements and risks. The requirement and risk having highest priority factor are tested first. Then priority of requirements and risks are calculated by assigning some weight to each requirement and risk. They have conducted an experiment by taking an example of Word Counter Software in which the frequency of words is calculated.

Samanta and Kundu (2008) developed a methodology to produce TCs using UML activity diagram. Here, the authors have followed three steps for generating the test cases. First, an activity diagram is constructed with all required test information. In the second step, an activity graph was developed from the AD and lastly test cases are produced from the graph by traversing all the nodes of the graph. Here, the authors have discussed different types of branch coverage and defects.

Huang et al. (2012) discussed a technique for cost-cognizent test case prioritization. The test cases prioritization is done on the basis of historical records. As per the proposed framework, the documented records were collected from current regression testing. After that, the authors had developed a new GA to detect highly efficient order of test cases. Prioritization was performed on the basis of historical data to provide high test effectiveness during the testing process.

Khalilian et al. (2012) discussed the enhancement of test case prioritization technique by using performance data of the historical test cases. In that technique, the author have considered an equation for test case prioritization. The equation directly computes the priority of each test case by using historical information of test case. But, they have proposed a new prioritization equation with variable coefficients gained according to the available historical performance data. The historical performance data of a program acts as a feedback from the previous test sessions. Finally, they have compared the proposed approach with their previous technique by measuring the effectiveness using APFD metric.

Shahid and Ibrahim (2014) presented a new algorithm for test case prioritization that is based on the code coverage of the test cases. In this paper, the authors introduced a new regression test case priority technique that arranges the selected test cases based on their code coverage percentage. The test cases which cover more code are selected at the top of the list. It is observed that the proposed approach reduce the time to find faults earlier. They also proposed a new algorithm for test case prioritization and implemented their algorithm using a case study of On-Board Auto-mobile (OBA) and found that the test cases which covers more methods have more chances to detect faults earlier.

4 Proposed framework

Here, we have discussed a heuristic technique for test case prioritization and the architecture of the proposed framework is shown in Fig. 1. In the the first step, we collect the modified requirements and model the system with new requirements. Since, we carry out model based testing, so we construct the system model by using Unified Modelling Language (UML) diagrams. In this research 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” interchangeably throughout the paper. 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 in terms of an adjacency matrix. 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

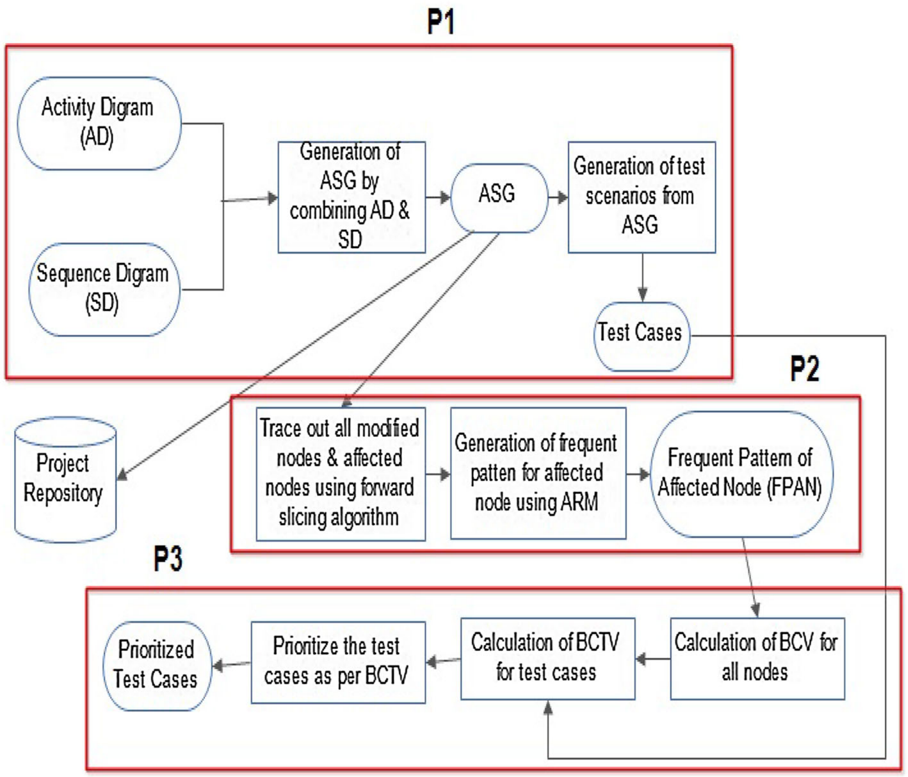

P1
Activity Digram (AD)
Sequence Digram (SD)
Generation of ASG by combining AD & SD
ASG
Generation of test scenarios from ASG
Test Cases
P2
Project Repository
Trace out all modified nodes & affected nodes using forward slicing algorithm
Generation of frequent patten for affected node using ARM
Frequent Pattern of Affected Node (FPAN)
P3
Prioritized Test Cases
Prioritize the test cases as per BCTV
Calculation of BCTV for test cases
Calculation of BCV for all nodes

Fig. 1 Architecture of the proposed approach

derived itemset is known as frequent itemset. Similarly, the CV of each frequent itemset is determined and also compared with threshold CV. Then, the generated final pattern is known as *Frequent Pattern of Affected Node (FPAN)*.

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.

In this proposed work, we have considered multiple number of modified nodes and corresponding affected nodes. These nodes are represented in a adjacency matrix. The frequent pattern is developed by using ARM taking the modified and affected nodes as input. We have considered different support values and confidence values to get an efficient frequent pattern.

The detailed procedure to execute the proposed framework for TCP using ARM is given below:

Let us consider, the AD and SD for the given system constructed earlier. We have done some modification to the existing system. As per user demand, we have added and modified few requirements in the existing system. Now, we discuss the steps of our proposed approach:

Step-1: Construct the Modified Activity and Sequence Diagrams

Construct the modified AD and SD by adding the new features to the old version of the system.

Step-2: Construct the Activity Sequence Graph (ASG) from Activity Diagram and Sequence Diagram

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 Diagram (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.

Step-3: Generate test scenarios by traversing the Activity Sequence Graph

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.

Step-4: Generate graph data and observation data from ASG

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 Graph Data (GD). Simultaneously, collect the information from the domain experts based on their observation which is referred to as observation data, during the execution of the application and store them in the project repository. This data is termed as Observation Data (OD).

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 $(N_1, N_2, N_3, ....)$;
7:     AN⟵ $\{N_1, N_2, N_3, .......N_m\}$; // 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 the activity sequence graph of the given system 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 (Coremen et al. 2010) and are stored in an array and treated as the set of affected nodes.

Step-5: Generate frequent pattern using association rule mining (ARM)

Next, the frequent pattern of nodes are developed by applying ARM on the project repository. The project repository contains both GD and OD . Identify specific support and confidence value for the mining operation. The frequent pattern is generated as per the confidence value.

Step-6: Determine BCV of all node

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

$$BCV(N) = \frac{Number\ of\ times\ the\ node\ encountered}{Total\ no.\ of\ nodes\ being\ affected} \quad (1)$$

Step-7: Generate prioritized test cases using BCTV

TCP is performed by using the BCTV of each test case. The BCTV of each test case is the summation of BCV of all nodes executed by the test case.

5 Case study: Hospital Management System

To explain the detail functioning of the proposed framework, we have used a case study of Hospital Management System (HMS). Hospital Management System is a software which automates the activities in a hospital such as patient registration, visitor interaction, fixing appointment with medical representatives, revert to phone queries etc. Patient registration includes checking the schedule of all doctors, taking appointment of doctor, admission of patient into ward, undergo operation, conduct pathological tests and update the required tests and reports etc. Now, the steps of our proposed framework is explained in detail by using the given case study.

- *Construct the Modified AD and SD*
  First, we construct the modified AD and SD for HMS (given in Figs. 2 and 3) with the new features of the system.
- *Construct the activity sequence graph (ASG) from AD and SD*
  Now, both the diagrams are converted into a model dependency graph called ASG. This graph is developed by combining the features and functionalities of AD and SD. For example, the function of Do operation of patient and operate() is similar. For that reason we have combined these two activities as a single node i.e. A23, S13 in the activity sequence graph. During the conversion, each node in the ASG represents the activity (from AD) along with the message transmitted (from SD) related to the activity. The activity flow is represented as edges between two nodes in the ASG (given in Fig. 4).
- *Generate TCs from activity sequence graph using graph traversal algorithm*
  Now, the activity sequence graph of HMS (given in

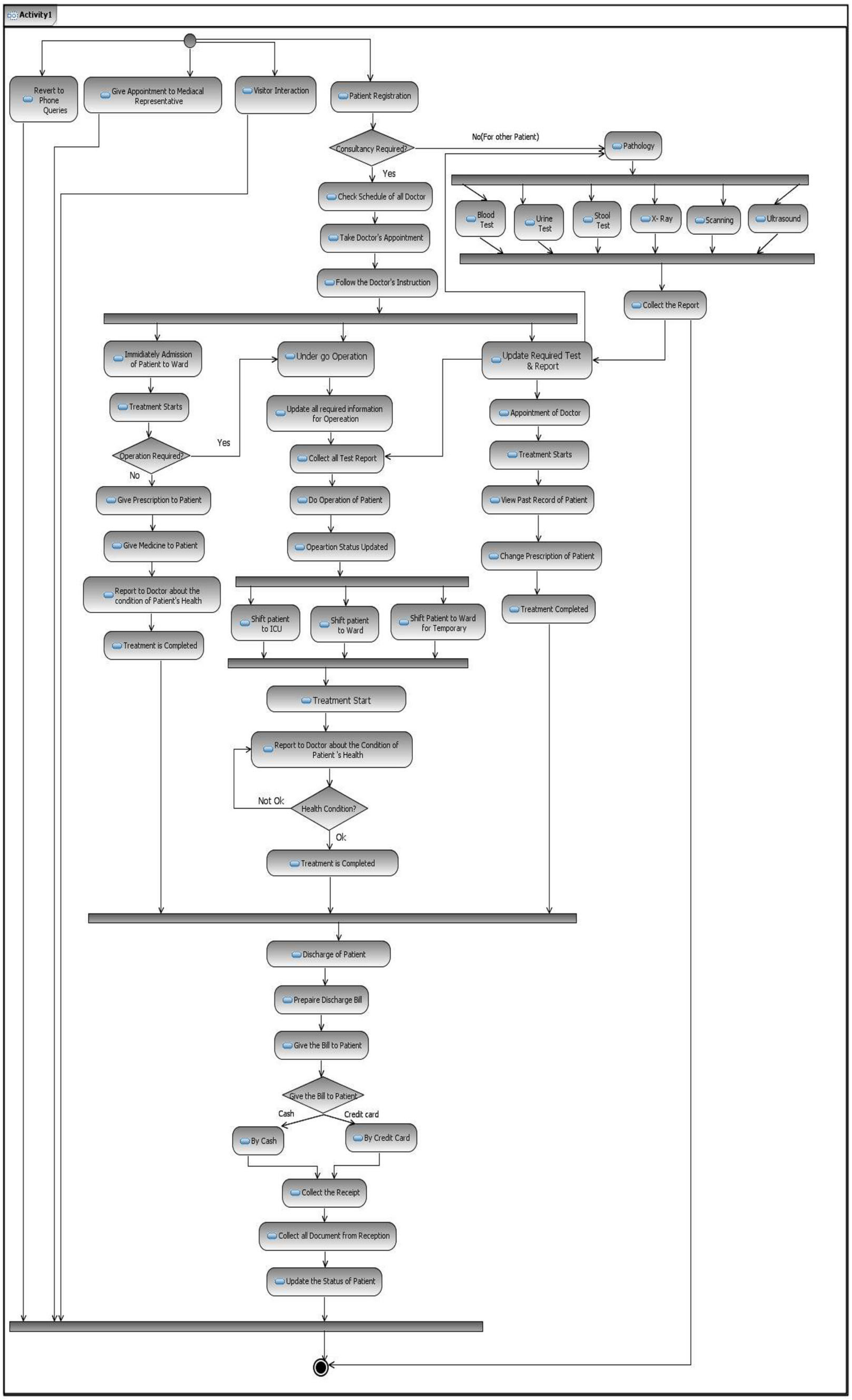


Fig. 2 Activity diagram of Hospital Management System

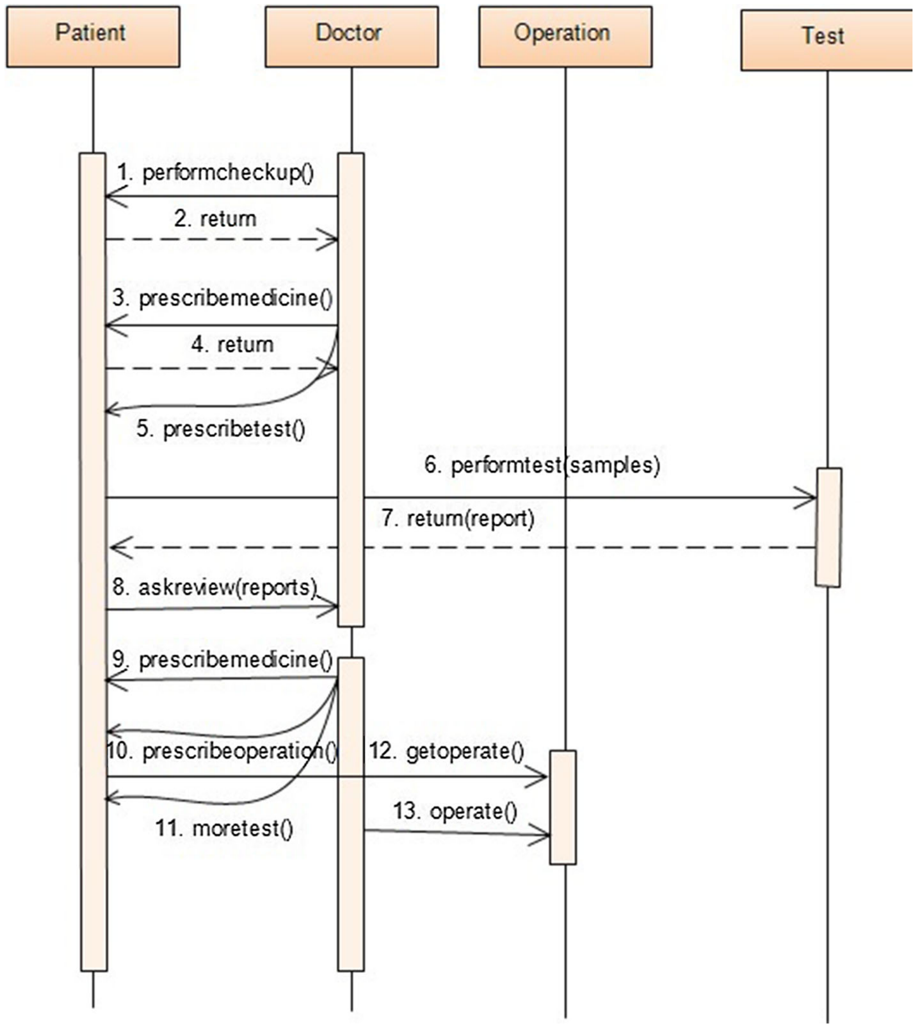


Fig. 3 Sequence diagram of Hospital Management System for giving prescription to patient

Fig. 4) is used for test scenarios generation by applying graph traversal algorithm. The graph generates 31 test scenarios and these test scenarios represent single paths in the activity sequence graph. The generated test scenarios are considered as TCs and are given in Table 2. Further the test scenarios are used for prioritization process after finding the frequent pattern of the modified nodes present in the activity sequence graph.

– *Generate graph data (GD) and observation data (OD) from activity sequence diagram*
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). Then, the modified information is saved in the project repository. The information that are collected from activity sequence graph are termed as Graph Data (GD). In the new version of project, we have considered five types of changes which are represented as C1, C2, C3, C4 and C5. These sets of nodes are called *itemset*. At the same time, Observation Data (OD) are collected from domain experts by observing the projects during the execution time. GD and OD for the given project are shown in Fig. 5 which are used to generate the frequent pattern using association rule mining. The modified nodes and the corresponding affected nodes with respect to different types of changes are given in Table 3. The database schema for the project repository contains all information about the developed application like Project-ID, Project-Name, Project-Type, Total-No-of-Nodes, Total-No-of-Modified-Nodes, Total-No-of-Affected-Node.

– *Generate frequent pattern using association rule mining (ARM)*
This phase gives a detailed description of generation of frequent pattern. After storing GD and OD in the project repository, the concept of data mining is used to generate the frequent pattern. There are different

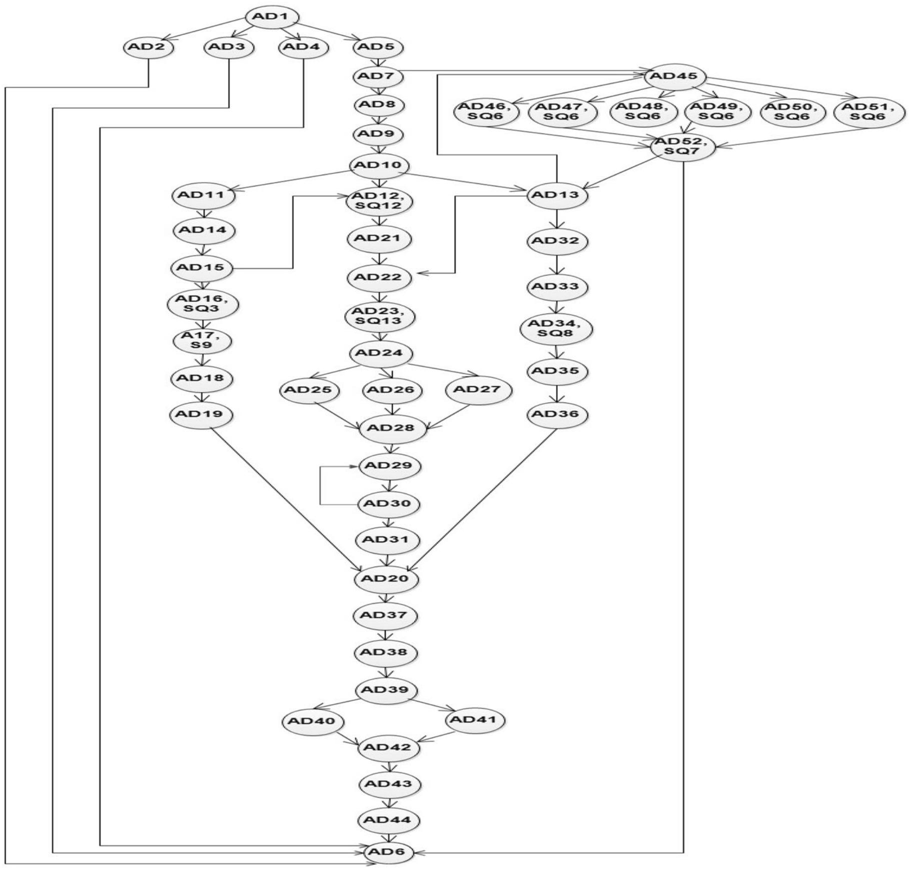


Fig. 4 ASG of Hospital Management System

methodologies available to find out frequent patterns like association rule, apriori algorithm, FP growth etc. (Han and Kamber 2010). We have considered association rule mining to generate frequent patterns. To generate frequent pattern, computation of two parameters are required i.e. *support* and *confidence* values. In the first step, SV of individual and combination of items is calculated from the whole itemset. The final SV is compared with user-specified *minimum or threshold SV* to generate frequent itemset. This frequent itemset is used for calculation of confidence value and compared with the user-specified *minimum or threshold CV*. The developed pattern is called Frequent Pattern of Affected Node (FPAN). Here, we have implemented the itemsets in MATLAB R2012a to produce the frequent pattern by taking *minimum support* value as 40% and *minimum confidence* value as 80%. The FPAN matrix from the implementation is shown in Fig. 6. The node sequence (FPAN) found from the matrix is given as:

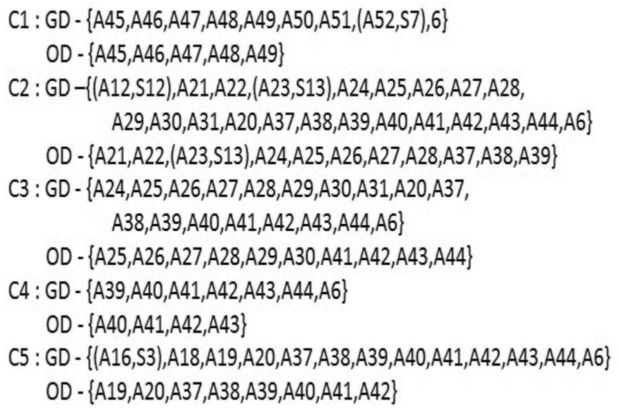
FPAN = {AD17, AD18, AD9, AD20, AD21, AD22, AD23, AD24, AD25, AD26, AD27, AD28, AD29, AD30, AD31, AD37, AD38, AD39, AD40, AD41, AD42, AD43,AD44, AD46, AD47, AD48, AD49, AD50, AD51, AD52}

– *Calculate the BCV of all nodes*
In this phase, BCV of all nodes present in the FPAN is determined by using the formula given in Eq. 1. The BCV of each nodes present in the pattern (i.e. FPAN) is given in Table 4. Then, we calculate the BCTV for each test cases, by taking the summation of business criticality values of all nodes executed by the TCs.
For example, BCTV for test case TC4 is calculated as follows:
BCTV of TC4 = 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0.033 + 0.033 + 0.033 + 0.1 + 0.1 + 0.1 + 0.133 + 0.133 + 0.133 + 0.133 + 0.133 + 0 = 1.064

– *Generate prioritized test scenarios using BCTV*
Here, prioritization of test cases is done using BCTV. The BCTV of a test case is determined by taking the summation of BCV of each node present in FPAN. If the node is present in the FPAN, then the value is retrieved form Table 4, otherwise the value is zero. The BCTV of all test cases are given in Table 5. The TCs are prioritized as per BCTV. The test case containing higher business criticality test value is considered as

Table 2 Generated test cases for HMS

SL no. (path)	Test case id	Independent path/Test Scenario
1	TC1	AD1 → AD2 → AD6
2	TC2	AD1 → AD3 → AD6
3	TC3	AD1 → AD4 → AD6
4	TC4	AD1 → AD5 → AD7 → AD8 → AD9 → AD10 → AD11 →AD14 → AD15 → AD16, SQ3 → AD17, SQ9 → AD18 → AD19 → AD20 → AD37→AD38 → AD39→AD40 → AD42→AD43 → AD44 → AD6
5	TC5	AD1 → AD5 → AD7 → AD8→ AD9 → AD10→AD11 → AD14→AD15 → AD16, SQ3→AD17, SQ9 → AD18 → AD19 → AD20 → AD37 → AD38 → AD39 → AD41 → AD42 → AD43 → AD44 → AD6
6	TC6	AD1 → AD5→AD7 → AD8 → AD9 → AD10 → AD11 → AD14 → AD15 → AD12, SQ12 → AD21 → AD22 → AD23, SQ13 → AD24 → AD25 → AD28 →AD29 → AD30 → AD31 → AD20 → AD37 → AD38 → AD39 → AD40 → AD42 → AD43 → AD44 → AD6
7	TC7	AD1 → AD5 → AD7 → AD8 → AD9 → AD10 → AD11 → AD14 → AD15 → AD12, SQ12 → AD21 → AD22 → AD23, SQ13 → AD24 → AD25 → AD28 →AD29 → AD30 → AD31 → AD20→ AD37 → AD38 → AD39 → AD41 → AD42 → AD43→AD44 → AD6
8	TC8	AD1 → AD5 → AD7 → AD8 → AD9 → AD10 → AD11 → AD14 → AD15 → AD12, SQ12 → AD21 → AD22 → AD23, SQ13 → AD24 → AD26 → AD28 →AD29 → AD30 → AD31 → AD20→ AD37 → AD38 → AD39 →AD40 → AD42 → AD43 → AD44 → AD6
9	TC9	AD1 → AD5→AD7 → AD8→AD9 → AD10 → AD11 → AD14 → AD15 → AD12, SQ12 → AD21 → AD22 → AD23, SQ13 → AD24 → AD26 → AD28 →AD29 → AD30 → AD31 → AD20→ AD37 → AD38 → AD39 → AD41→AD42 → AD43 → AD44 → AD6
10	TC10	AD1 → AD5→AD7 → AD8 → AD9 → AD10→ AD11 → AD14 → AD15 → AD12, SQ12 → AD21 → AD22 →AD23, SQ13 → AD24→AD27 → AD28 → AD29 → AD30 → AD31 → AD20 → AD37 → AD38→AD39 → AD40 →AD42 → AD43 → AD44 → AD6
11	TC11	AD1 → AD5 → AD7 → AD8 → AD9 → AD10 → AD11 → AD14 → AD15 → AD12, SQ12 → AD21 → AD22 → AD23, SQ13 → AD24 → AD27 → AD28 → AD29 → AD30 → AD31 → AD20 → AD37 → AD38 → AD39 → AD41 → AD42 → AD43 →AD44 → AD6
12	TC12	AD1 → AD5 → AD7 → AD8 → AD9 → AD10 → AD12, SQ12 → AD21 → AD22 → AD23, SQ13 → AD24 → AD25 → AD28 → AD29 → AD30 → AD31 → AD20 → AD37 → AD38 → AD39 → AD40 → AD42 → AD43 → AD44 → AD6
13	TC13	AD1 → AD5→AD7 → AD8 → AD9 → AD10 → AD12, SQ12 → AD21 → AD22 → AD23, SQ13 → AD24 → AD25 → AD28 → AD29 → AD30 → AD31 → AD20 → AD37 → AD38 → AD39 → AD41 → AD42 → AD43 →AD44 → AD6
14	TC14	AD1 → AD5 → AD7 → AD8 → AD9 → AD10 → AD12, SQ12 → AD21 → AD22 → AD23, SQ13 → AD24 → AD26 → AD28 → AD29 → AD30 → AD31 → AD20 → AD37 → AD38 → AD39 → AD40 → AD42 → AD43→AD44 → AD6
15	TC15	AD1 → AD5 → AD7 → AD8 → AD9 → AD10 → AD12, SQ12 → AD21 → AD22 → AD23, SQ13 → AD24 → AD26 → AD28 → AD29 → AD30 → AD31 → AD20 → AD37 → AD38 → AD39 → AD41 → AD42 → AD43 → AD44 → AD6
16	TC16	AD1 → AD5 → AD7 → AD8 → AD9 → AD10 → AD12, SQ12 → AD21 → AD22 → AD23, SQ13 → AD24 → AD27 → AD28 → AD29 → AD30 → AD31 → AD20 → AD37 → AD38 → AD39 → AD40 → AD42 → AD43 → AD44 → AD6
17	TC17	AD1 → AD5 → AD7 → AD8 → AD9 → AD10 → AD12, SQ12 → AD21 → AD22 → AD23, SQ13 → AD24 → AD27 → AD28 → AD29 → AD30 → AD31 → AD20 → AD37 → AD38 → AD39 → AD41 → AD42 → AD43 → AD44 → AD6
18	TC18	AD1 → AD5 → AD7 → AD8→ AD9 → AD10 → AD13 → AD32 → AD33 → AD34, SQ8 → AD35 → AD36 → AD20 → AD37 → A3D8 → AD39 → AD40 → AD42 → AD43 → AD44 → AD6
19	TC19	AD1 → AD5 → AD7 → AD8 → AD9 → AD10 → AD13 → AD32 → AD33 → AD34, SQ8 → AD35 → AD36 → AD20 → AD37 → A3D8 → AD39 → AD41 → AD42 → AD43 → AD44 → AD6
20	TC20	AD1 → AD5 → AD7 → AD8 → AD9 → AD10 → AD13 → AD22 →AD23, SQ13 → AD24 → AD25 → AD28→AD29 → AD30 → AD31 → AD20 → AD37 → AD38 → AD39 → AD40 → AD42 → AD43 → AD44 → AD6
21	TC21	AD1 → AD5 → AD7 → AD8 → AD9 → AD10 → AD13 → AD22 → AD23, SQ13 → AD24 → AD25 → AD28 →AD29 → AD30 → AD31 → AD20 → AD37 → AD38 → AD39 → AD41→AD42 → AD43 → AD44 → AD6
22	TC22	AD1 → AD5 → AD7 → AD8 → AD9 → AD10 → AD13 → AD22 → AD23, SQ13 → AD24 → AD26 → AD28 →AD29 → AD30 → AD31 → AD20 → AD37 → AD38 → AD39 → AD40 → AD42 → AD43 → AD44→ AD6
23	TC23	AD1 → AD5→AD7 → AD8→ AD9 → AD10 → AD13 → AD22 → AD23, SQ13 → AD24 → AD26 → AD28→AD29 → AD30 →AD31 → AD20 → AD37 → AD38 → AD39 → AD41→AD42 → AD43 → AD44 → AD6
24	TC24	AD1 → AD5 → AD7 → AD8 → AD9 → AD10 → AD13 → AD22 → AD23, SQ13 → AD24 → AD27 → AD28 →AD29 → AD30 → AD31 → AD20 → AD37 → AD38 → AD39 → AD40 → AD42 → AD43 → A44→A6
25	TC25	AD1 → AD5 → AD7 → AD8 → AD9 → AD10 → AD13 → AD22 → AD23, SQ13 → AD24 → AD27 → AD28 → AD29 → AD30 → AD31 → AD20 → AD37 → AD38 → AD39 → AD41 → AD42 → AD43 → A44 → A6
26	TC26	AD1 → AD5 → AD7 → AD45 → AD46 → AD52, SQ7 → AD6
27	TC27	AD1 → AD5 → AD7 → AD45 → AD47 → AD52, SQ7 → AD6
28	TC28	AD1 → AD5 → AD7 → AD45 → AD48 → AD52, SQ7 → AD6
29	TC29	AD1 → AD5 → AD7 → AD45 → AD49 → AD52, SQ7 → AD6
30	TC30	AD1 → AD5→AD7 → AD45 → AD50 → AD52, SQ7 → AD6
31	TC1	AD1 → AD5→AD7 → AD45 → AD51 → AD52, SQ7 → AD6



C1 : GD - {A45,A46,A47,A48,A49,A50,A51,(A52,S7),6}
OD - {A45,A46,A47,A48,A49}
C2 : GD –{(A12,S12),A21,A22,(A23,S13),A24,A25,A26,A27,A28,
A29,A30,A31,A20,A37,A38,A39,A40,A41,A42,A43,A44,A6}
OD - {A21,A22,(A23,S13),A24,A25,A26,A27,A28,A37,A38,A39}
C3 : GD - {A24,A25,A26,A27,A28,A29,A30,A31,A20,A37,
A38,A39,A40,A41,A42,A43,A44,A6}
OD - {A25,A26,A27,A28,A29,A30,A41,A42,A43,A44}
C4 : GD - {A39,A40,A41,A42,A43,A44,A6}
OD - {A40,A41,A42,A43}
C5 : GD - {(A16,S3),A18,A19,A20,A37,A38,A39,A40,A41,A42,A43,A44,A6}
OD - {A19,A20,A37,A38,A39,A40,A41,A42}

Fig. 5 Graph data and observation data of Hospital Management System

higher priority than other test cases. 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: {TC6, TC7, TC8, TC9, TC10, TC11, TC12, TC13, TC14, TC15, TC16, TC17, TC20, TC21, TC22, TC23, TC24, TC25, TC4, TC5, TC18, TC19, TC30, TC31, TC1, TC2 and TC3}.

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 follows (Chauhan 2016):

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

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 like message dependency fault, inter/intra activity dependency fault, control dependency fault etc. (given in Table 6). Table 7 represents the total number of faults or defects detected by each TC.

For the prioritized test suite,

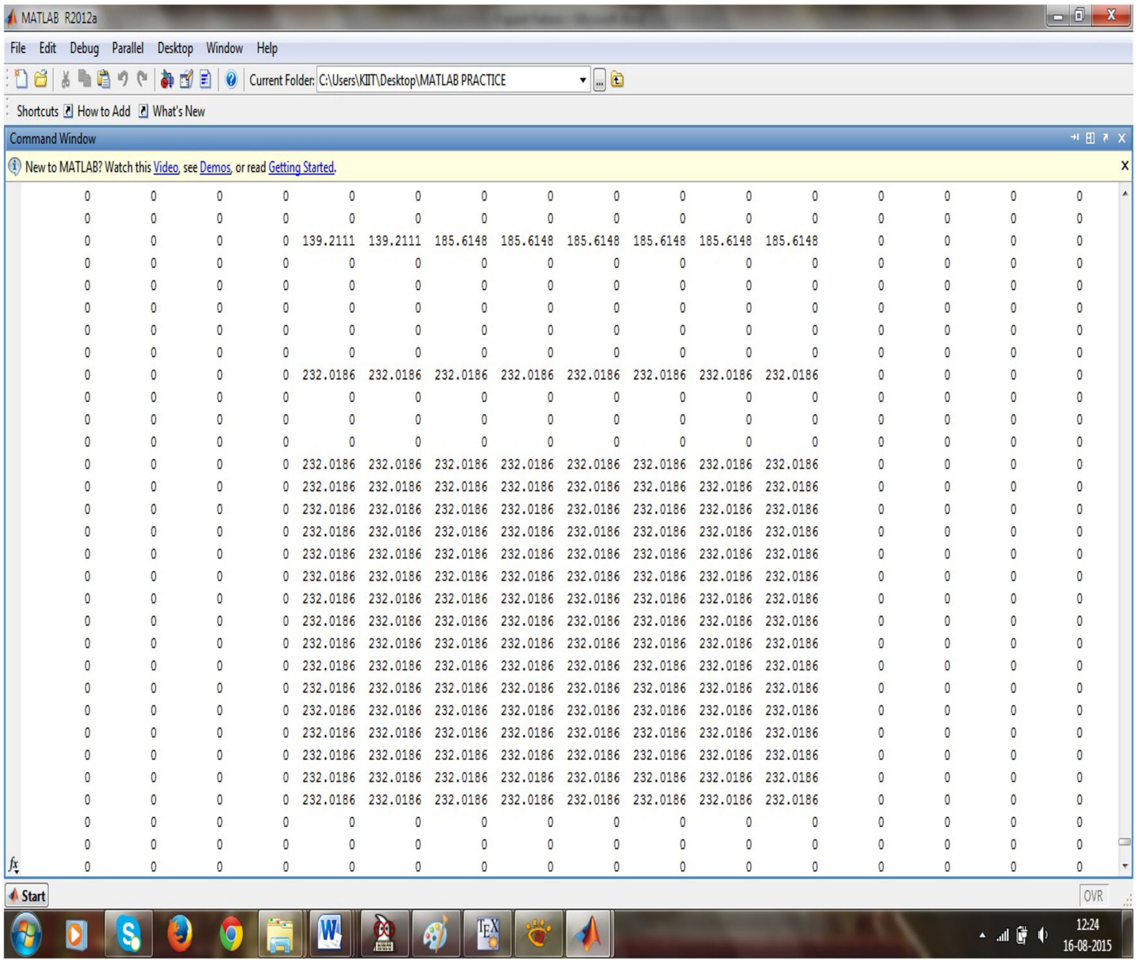
$$\begin{aligned}\text{APFD} &= 1 - \frac{1+7+16+24+2+11+4+12+6+2+11+7+5+22+20+6+4+7+6+6}{20 \times 31} + \frac{1}{2 \times 31} \\ &= 1 - 0.2887 + 0.0161 = 0.7274\end{aligned}$$

For the non-prioritized test suite,

$$\begin{aligned}\text{APFD} &= 1 - \frac{1+2+11+17+3+6+3+7+1+4+6+2+18+15+13+3+2+2+1+1}{20 \times 31} + \frac{1}{2 \times 31} \\ &= 1 - 0.1903 + 0.0161 = 0.8258\end{aligned}$$

Table 3 Modified nodes and affected nodes derived from Fig. 4

Changes	Type of change	Modified nodes	Affected nodes
C1	Addition of new requirement	AD45	AD45, AD46, AD47, AD8, AD49, AD50, AD51, (AD52, SD7), AD6
C2	Modification in the existing requirement	AD12, SQ12	(AD12, SQ12), AD21, AD22, (AD23, SQ13), AD24, AD25, AD26, AD27, AD28, AD29, AD30, AD31, AD20, AD37, AD38, AD39, AD40, AD41, AD42, AD43, AD44, AD6
C3	Addition of new requirement	AD24	AD24, AD25, AD26,AD27, AD28, AD29, AD30, AD31, AD20,AD37, AD38, AD39, AD40, AD41, AD42, AD43, AD44, AD6
C4	Addition of new requirement	AD39	AD39, AD40, AD41, AD42, AD43, AD44, AD6
C5	Changes in the existing requirement	AD16, SQ3	(AD16, SQ3), AD18, AD19, AD20, AD37, AD38, AD39, AD40, AD41, AD42, AD43, AD44, AD6


MATLAB R2012a
File Edit Debug Parallel Desktop Window Help
Current Folder: C:\Users\KIIT\Desktop\MATLAB PRACTICE
Shortcuts How to Add What's New
Command Window
Start
OVR
12:24
16-08-2015

New to MATLAB? Watch this Video, see Demos, or read Getting Started.

```
         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0
         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0
         0         0         0         0  139.2111  139.2111  185.6148  185.6148  185.6148  185.6148  185.6148  185.6148         0         0         0         0
         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0
         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0
         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0
         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0
         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0
         0         0         0         0  232.0186  232.0186  232.0186  232.0186  232.0186  232.0186  232.0186  232.0186         0         0         0         0
         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0
         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0
         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0
         0         0         0         0  232.0186  232.0186  232.0186  232.0186  232.0186  232.0186  232.0186  232.0186         0         0         0         0
         0         0         0         0  232.0186  232.0186  232.0186  232.0186  232.0186  232.0186  232.0186  232.0186         0         0         0         0
         0         0         0         0  232.0186  232.0186  232.0186  232.0186  232.0186  232.0186  232.0186  232.0186         0         0         0         0
         0         0         0         0  232.0186  232.0186  232.0186  232.0186  232.0186  232.0186  232.0186  232.0186         0         0         0         0
         0         0         0         0  232.0186  232.0186  232.0186  232.0186  232.0186  232.0186  232.0186  232.0186         0         0         0         0
         0         0         0         0  232.0186  232.0186  232.0186  232.0186  232.0186  232.0186  232.0186  232.0186         0         0         0         0
         0         0         0         0  232.0186  232.0186  232.0186  232.0186  232.0186  232.0186  232.0186  232.0186         0         0         0         0
         0         0         0         0  232.0186  232.0186  232.0186  232.0186  232.0186  232.0186  232.0186  232.0186         0         0         0         0
         0         0         0         0  232.0186  232.0186  232.0186  232.0186  232.0186  232.0186  232.0186  232.0186         0         0         0         0
         0         0         0         0  232.0186  232.0186  232.0186  232.0186  232.0186  232.0186  232.0186  232.0186         0         0         0         0
         0         0         0         0  232.0186  232.0186  232.0186  232.0186  232.0186  232.0186  232.0186  232.0186         0         0         0         0
         0         0         0         0  232.0186  232.0186  232.0186  232.0186  232.0186  232.0186  232.0186  232.0186         0         0         0         0
         0         0         0         0  232.0186  232.0186  232.0186  232.0186  232.0186  232.0186  232.0186  232.0186         0         0         0         0
         0         0         0         0  232.0186  232.0186  232.0186  232.0186  232.0186  232.0186  232.0186  232.0186         0         0         0         0
         0         0         0         0  232.0186  232.0186  232.0186  232.0186  232.0186  232.0186  232.0186  232.0186         0         0         0         0
         0         0         0         0  232.0186  232.0186  232.0186  232.0186  232.0186  232.0186  232.0186  232.0186         0         0         0         0
         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0
         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0
         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0         0
```

Fig. 6 Frequent pattern matrix of Hospital Management System

Table 4 Business criticality value of all nodes present in FPAN

Affected nodes	No. of times the node is encountered	BCV of nodes	Affected nodes	No. of times the node is encountered	BCV of nodes
AD17, SQ9	1	0.033	AD37	3	0.1
AD18	1	0.033	AD38	3	0.1
AD19	1	0.033	AD39	4	0.133
AQ20	3	0.1	AD40	4	0.133
AD21	1	0.033	AQ41	4	0.133
AD22	1	0.033	AD42	4	0.133
AD23, SQ13	1	0.033	AD43	4	0.133
AD24	2	0.066	AD44	4	0.133
AD25	2	0.066	AD46, SQ6	1	0.033
AD26	2	0.066	AD47, SQ6	1	0.033
AD27	2	0.066	AD48, SQ6	1	0.033
AD28	2	0.066	AD49, SQ6	1	0.033
AD29	2	0.066	AD50, SQ6	1	0.033
AD30	2	0.066	AD51, SQ6	1	0.033
AD31	1	0.133	AD52, SQ7	1	0.033

Table 5 BCTV of all test cases

Sl no.	Test scenario Ids	BCTV of test scenarios	Priority of test scenarios
1	TC1, TC2, TC3	0	6
2	TC4, TC5	1.064	3
3	TC6, TC7, TC8, TC9, TC10, TC11, TC12, TC12, TC13, TC13, TC14, TC15, TC16, TC17	1.427	1
4	TC18, TC19	0.965	4
5	TC20, TC21, TC22, TC23, TC24, TC25	1.394	2
6	TC26, TC27, TC28, TC29, TC30, TC31	0.066	5

Table 6 Different types of faults

Fault types	Detected Faults
Inter-activity dependency fault	FT1, FT5, FT10, FT15
Intra-activity dependency fault	FT6, FT7, FT13, FT18, FT20
Synchronization fault	FT3, FT8, FT11, FT12
Control dependency fault	FT5, FT9, FT16, FT17
Message dependency fault	FT2, FT4, FT10, FT19
Operational fault	FT1, FT5, FT14, FT15
Fault in loop	FT5, FT16, FT17, FT19, FT20

APFD value for the given prioritized test suite is calculated to be 0.8258 and for the non-prioritized test suite is 0.7274. 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 7 shows the APFD values for the prioritized and non-prioritized TCs for our case study. To further validate our approach, we have considered four other case studies i.e. Library Management System (LMS), Shopping Mall Management System (SMMS), On-line Examination System (OES), Online-Ticket Booking System (OTB). The respective APFD values of prioritized and non-prioritized TCs for these case studies are shown in Table 8 and Fig 8. It is observed that the APFD value of prioritized TCs is greater than the APFD value of non-prioritized TCs.

6 Comparison with related work

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. In the context of model based testing, (Muthusamy and Seetharaman 2014) and (Indumathi and Selvamani 2015) proposed some techniques based on dependency structure prioritization. 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 behaviour 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 (Acharya et al. 2015). The APFD value is also increased in this case. But, all the MBTP techniques are using single UML diagram for system modeling. So, the

Table 7 Total number of faults detected by each test case in HMS

	FT1	FT2	FT3	FT4	FT5	FT6	FT7	FT8	FT9	FT10	FT11	FT12	FT13	FT14	FT15	FT16	FT17	FT18	FT19	FT20
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		×								×										

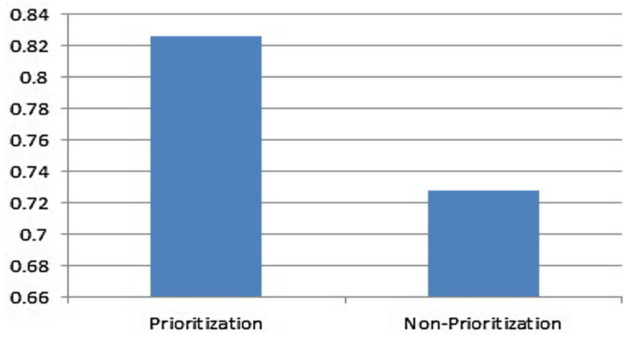


Fig. 7 APFD values of prioritized and non-prioritized test cases of our HMS case study

Table 8 APFD values for different case studies

Sl no.	Project name	Number of test cases	APFD value (prioritized)	APFD value (non-prioritized)
1	Hospital Management System (HMS)	31	0.8258	0.7274
2	Library Management System (LMS)	37	0.8576	0.8025
3	Shopping Mall Management System (SMMS)	28	0.7786	0.7298
4	On-line Examination System (OES)	22	0.7339	0.6989
5	On-line Ticket Booking (OTB)	18	0.7020	0.6563

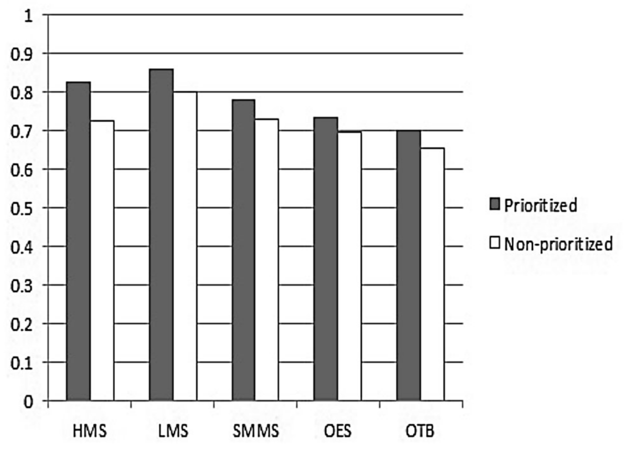


Fig. 8 APFD values of prioritized and non-prioritized test cases for different case studies

type of faults detected by any test case is limited. So, we have taken combination of UML diagrams i.e. AD and SD for system modelling. This will help in identifying different types of faults. In our approach, the prioritized test sequence is capable of detecting different types of faults i.e. message dependency fault, fault in loop, synchronization fault etc. and the APFD value is also increased accordingly. In this work, we have only considered the functionalities of the system while not taking into account the non-functional aspects. We have compared our approach with different related approaches considering the same case studies. The comparison results with some related works are summarized in Table 9. The comparison result is also represented by a graph shown in Fig. 9. From Table 9 and Fig. 9, it can be observed that our proposed approach provides higher quality results than that of others.

7 Conclusion and future work

In this research work, we have presented a technique for test case prioritization using UML AD, SD and association rule mining (ARM). The modified system is modelled using UML AD and SD. Then, activity sequence graph is constructed by combining both the diagrams and TCs are developed from that graph. Simultaneously the detail of the graph is stored in the project repository. Then, all the modified nodes and respective affected nodes are traced by using forward slicing algorithm. After that, a pattern called FPAN is developed by using ARM. The proposed framework is explained using a case study named Hospital Management System (HMS). To verify the efficiency and effectiveness of this approach, we have also experimented our approach with different support and confidence values to observe the behaviour of the frequent patterns during mining the graph data and observation data. In addition to this, another major criteria i.e. Business Criticality Value is added for TCP. The proposed framework for TCP is also applied on several case studies and found to be very useful in early fault detection. During the whole process, we haven't considered the non-functional aspects of the system under test. 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.

Table 9 APFD values of different approaches

Author's name	HMS	LMS	SMMS	OES	OTB
Non-prioritized case	0.7274	0.8025	0.7298	0.6989	0.6563
Khandai et al. (2011)	0.6675	0.7654	0.6932	0.6158	0.6351
Muthusamy and Seetharaman (2014)	0.6492	0.6864	0.7318	0.6734	0.6018
Indumathi and Selvamani (2015)	0.7285	0.7591	0.7453	0.6982	0.6578
Acharya et al. (2015)	0.7763	0.7985	0.7521	0.7167	0.6861
Our proposal	0.8258	0.8576	0.7786	0.7339	0.7020

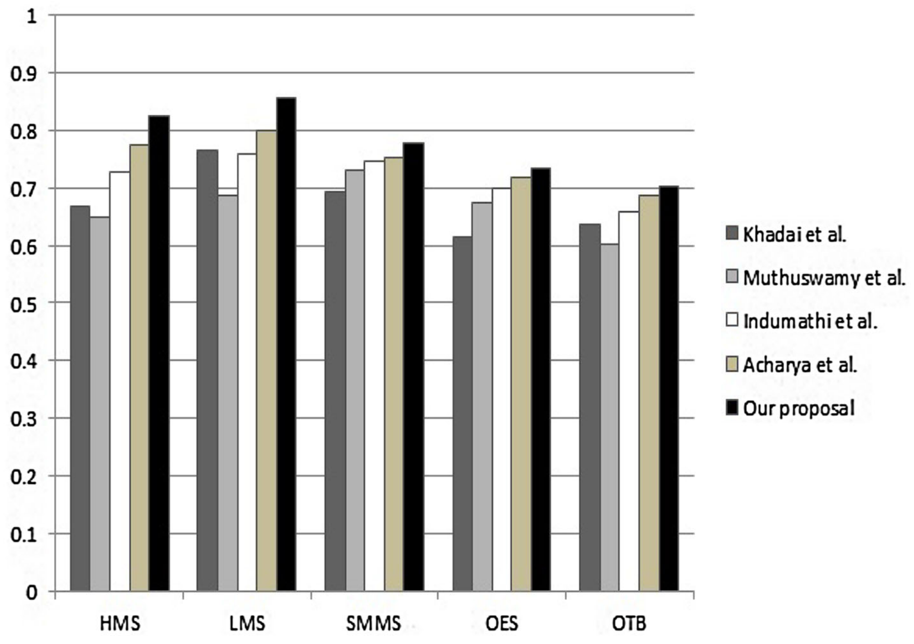


Fig. 9 Graph for APFD values of different approaches

References

1990. IEEE Standard Glossary of Software Engineering Terminology. www.ieeexplore.ieee.org/ IEEE Standard Glossary of Software Engineering Terminology

Acharya AA, Mahali P, Mohapatra DP (2015) Model based test case prioritization using association rule mining. Comput Intell Data Min 3:429–440

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

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

Chauhan N (2016) Software testing principles : practices, 2nd edn. Oxford University Press, New Delhi

Coremen TH, Leiserson CE, Rivest RL, Stein C (2010) Introduction to algorithms, 2nd edn. PHI Learning Private Limited, New Delhi

Garg D, Datta A, French T (2012) New test case prioritization strategies for regression testing of web applications. Int J Syst Assur Eng Manag 3(4):300–309

Han J, Kamber M (2010) Data mining: concepts and techniques. Morgan Kaufmann Publishers, 500 Sansome Street, Suite 400, San Francisco, CA 94111, 2nd edition

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

Huang Y-C, Peng K-i, Huang C-Y (2012) A history-based cost-cognizent test case prioritization in regression testing. J Syst Softw 85:626–637

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

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

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

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

Mall R (2014) Fundamental of software engineering, 4th edn. PHI Learning Private Limited, New Delhi

Mathur AP (2008) Foundations of software testing, 1st edn. Addison-Wesley Professional, Boston

Muthusamy T, Seetharaman K (2014) A new effective test case prioritization for regression testing based on prioritization algorithm. Int J Appl Inf Syst (IJAIS) 6(7):21–26

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

Rava M, Wan-Kadir WMN (2016) A review on prioritization techniques in regression testing. Int J Softw Eng Appl 10(1):221–232

Samanta D, Kundu D (2008) A novel approach to generate test cases from UML activity diagrams. J Object Technol 8(3):65–83

Sarma M, Mall R (2007) Automatic test case generation from UML models. In: Proceedings of 10th international conference on information technology, pp 196–201

Shahid M, Ibrahim S (2014) A new code based test case prioritization technique. Int J Softw Eng Appl 8(6):31–38

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

Swain SK, Mohapatra DP (2010) Test case generation from behavioural UML models. Int J Comput Appl 6(8):5–11

Tyagi M, Malhotra S (2015) An approach for test case prioritization based on three factors. Int J Inf Technol Comput Sci 4:79–86

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