IJCSE International Journal of Computer Sciences and Engineering Open Access
Research Paper Vol.-6, Issue-10, Oct 2018 E-ISSN: 2347-2693

Model-based Test Case Prioritization

S.S. Basa[1*], S.K. Swain[2], D.P. Mohapatra[3]

[1]Dept. of Computer Science, North Orissa University, Baripada, Odisha, India
[2]School of Computer Engineering, KIIT Deemed to be University, Bhubaneswar, Odisha, India
[3]Dept. of Computer Science & Engineering, NIT Rourkela, Rourkela, Odisha, India

[*]*Corresponding Author: santiswarup.basa@gmail.com, Tel.: +91-94396-58795*

Available online at: www.ijcseonline.org

Accepted: 16/Oct/2018, Published: 31/Oct/2018

***Abstract*-** The efficiency of Software testing can be improved by scheduling the test cases using test case prioritization technique (TCP). A novel test case prioritization approach is proposed to schedule the execution of test cases in testing process of software development. Our approach prioritizes the test cases generated from UML Sequence diagram. The major objective of our TCP approach is to achieve high rate of fault detection and test coverage. In this paper, an intermediate graph is created form UML sequence diagram to generate the message sequence paths. We calculate the weights of each node of the graph according to the affecting nodes using forward slice and edge using information flow model. 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. The result of our proposed approach is compared with the result of traditional approach using APFD for some selected software. Finally, our proposed prioritization approach is also compared with some available related work.

***Keywords*-**Test case prioritization, sequence diagram, message sequence path, forward slicing, information flow model

I. Introduction

Software has become an important item used in our daily life, starting from the small mobile applications to home applications, banks, medical, education, business etc. In every step of our life it is an indispensable requirement for each one of us. Testing of the present day software is a big challenge due to the complexity of present need. In the software development process software testing is an expensive phase as it takes longer time to produce a good quality and reliable software [2]. More than 50% of the produced software could not able to launch to the market due to lack of its proper testing [1].

Testing using code based design is a traditional approach from which the test cases will be generated after coding [3]. This method is very difficult and tedious. At the earlier stages of software development, testing can be done by using design documents such as Unified Modeling Language (UML). The test need not have to wait till the end of the product development. From the design document, testing can be done, so that early detection of the faults can be achieved. For this purpose, the diagram can be converted to an intermediate graph to generate and prioritize the test cases.

It is a challenge to achieve maximum throughput in software testing by uncovering the flaws from complex software. The quality of the software is measured as per the versatility of its use and the rigorous testing processes it goes across. Software testers continuously test the product by executing the test cases to find the bugs during software development. It is required by the testers to detect the faults as early as possible. Further, in the evolution process of software development, it is difficult to test all the test cases in a test suite as its size grows in subsequent evolutions. So to meet the goal like detecting faults early and to meet the resource constraints, the TCP (TCP) schedules the test cases in a suitable order for execution in testing or retesting. It increases chances of early finding of errors and improves efficiency of testing particularly in regression testing. Several traditional prioritization techniques focuses on rate of early requirement coverage criterion exercised by the test cases. In TCP, the major objective is to achieve high rate of fault detection and test coverage.

The rest parts of the paper are scheduled as: Some related basic concepts are presented in Section 2. The review of literatures is discussed in Section 3. Prioritization of test cases based on our approach is presented in Section 4. In section 5, the case study with result and analysis is elaborated. In section 6, we compare the results of our proposed approach with regard to traditional approach. Finally, Conclusion and future directions of our work are presented in Section 7.

II. Basic Concepts

We discuss UML sequence diagram and the different testing concepts used to understand our approach in this section. Then the basic concept of slicing is described which is used for prioritization of test cases.

A. Sequence Diagram

A sequence diagram SD = < OBJ, MSG > models the interaction of a set of objects. It consists of a set of objects OBJ and a sequence of messages MSG. Each message msg = < OBJ1, OBJ2 , M > ∈ MSG represents a sequential flow of message from one object OBJ1 to another object OBJ2. Sequential flow of control can be activated via several means. Figure 1 is an example of sequence diagram for an Account Checking use case.

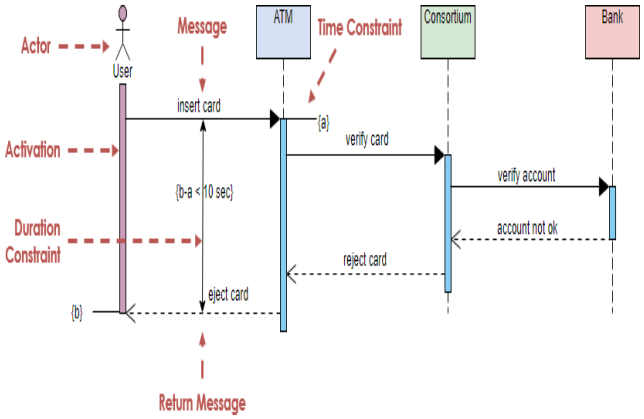


Figure 1. Example of UML Sequence diagram

B. Testing Concepts

Test Case: It is represented through [I, O, C] where I represents input and O represents expected output under a set of pre-conditions C. A tester determines whether a system under test executes as per the requirement specification or not using test cases. The process of generating test cases also helps in finding problems in the requirements or design specification.

Test Scenario: Test scenario represents the end to end functioning to check the process flow of the system under test. Test cases can be generated from test scenarios. One test scenario can have one or more test cases [4].

Message Flow graph: It is a directed graph, MFG= {N, D} where N is the set of nodes and D is the set of directed edges. Here N is the set of messages passed in sequence diagram and D denotes the edge. Edges of MFG called as dependency edges represent dependencies among nodes.

Message Flow Dependency Graph (MFDG): MFDG is constructed from MFG by adding parameter dependency edges among nodes representing the messages of sequence diagram.

Message sequence path coverage : Suppose a set of test cases TC is generated from MFG of a sequence diagram SD. Then the test cases achieve the message sequence path coverage if TC follows at least one message path in MFG.

C. Test Case Prioritization

Test case prioritization technique finds order of the test cases for the test execution with higher priority. So prioritized test cases detect the faults rapidly. This to show how rapidly a test suite detects faults. Prioritized test suites for regression testing saves time and cost of retesting. With the intention that more error prone parts of software are executed and detect faults with prioritized test cases [6]. During test execution, prioritized test cases will detect more errors if executed early. Rothermel presented a metric named APFD metric (Average Percentage of Faults Detected) to measure the performance of prioritization [7].APFD is used to calculate the weighted average percentage of errors uncovered during the running of the test suite [5]. Let TS be a test suite which contains q test cases, and let F represents p faults detected by TS. Let FS_i be the number of first test case in ordering TS′ of TS which detects fault i.

According to rothermel [7], the APFD for test suite TS′ is computed as follows :

$$APFD = 1 - \frac{TF_1 + TF_2 + ... + TF_m}{qp} + \frac{1}{2q} \quad (1)$$

where p represents the number of faults and q represents the total number of test cases.

D. Slicing

Originally, Weiser[8] has introduced the concept of program slicing technique to analyze the program by flow of data and controls for debugging purpose [9]. Program slicing technique separates out the set of statements which are affected portion of a program with concerning to a specific parameter [10-13]. Slicing criterion for program slice refers to a point of interest for which the slicing will be carried out. Slicing criterion is represented as (l,r), where l is the location number of statement point and r is the parameter that is being used or defined at l. A program slice contains the set of statements which are affected by the values defined at slicing criterion. There are different types of program slices for use in various applications.

Forward Slicing: A forward slice gives the information of statements affected by the parameter r defined at l [14]. A forward slice represent a slicing criterion <l,r>, search the statements those are affected by the parameter r computed at l.

Backward Slicing : Backward slicing finds the subset of statements those are affected by the value of a parameter at the statement defined in slicing criterion [15]. This can be

computed by moving backward from bottom to top over the program. It finds all the statements that have an effect on the value of a parameter at the point of interest. Hence, a static backward slice finds which statements are affected by the slicing criterion.

III. Related Work

In this Section, we present some of the existing prioritization approaches proposed by different researchers.

Kundu et al. [16] proposed a prioritization technique suitable for project planning using use case scenarios. They have considered cohesion and coupling to enhance the software quality and reliability. They have used single use case without considering the use case relationships. Scenario cost metric is calculated using scenario path of use case diagram. In their technique, they detected and compared the overlapping scenario sub path from other scenario path to get the analytical solution to be used in project management activities. They showed efficiency of their approach by an experiment.

Panigrahi et al. [17] presented their work for prioritization of test cases using a model-based TCP approach. The forward as well as backward slicing techniques are used to find the affected program elements using data and control dependencies. The affected elements of the used model are identified with test cases using forward slicing. Backward slicing is used to mark the model elements those are affected by test cases. Their approach is suitable specifically for object-oriented programs.

Sapna et al. [18] proposed a black-box testing technique using Steiner Tree algorithm to generate minimal test cases. They divided the nodes into terminal and non terminal nodes. The terminal nodes were the input to the Steiner Tree algorithm. Minimal paths were only found considering the edge weight. Then the test cases were generated from the generated test paths.

Panda et al. [19] focussed on schduling of test cases for execution on the basis of higher priority for regression testing. They presented a code based static technique by converting a program into Affected Slice Graph (ASG). Then, the coupling values of affected nodes are calculated. By adding those coupling values of nodes of a test path are used to order the test cases. The approach was experimented with mutation faults to show the fault-proneness of test cases. According to fault-proneness of test case, they prioritize the test cases. Lastly, they have compared their approach with other existing techniques for prioritization of test cases for the input program.

Gupta et al.[20] discussed a prioritization approach which improve the efficiency of regression testing by scheduling the test cases. They multiplied statement coverage with function calls which is used for ordering of test cases. Lastly, they analysed the efficiency of ordering of test cases using APFD metric.

Jeffrey Dennis et al.[21] Proposed a slice and requirement coverage based method to prioritize the test cases. To show the effectiveness of their approach, they compared the experimental results with traditional techniques. They showed how the outcomes of test cases are influenced by statement and branches using the relevant slices. The prioritized test cases detect early faults.

IV. Proposed test case prioritization technique

We first design a UML sequence diagram for particular use case of a specific system. Then we generate XMI code of the model. By analysing XMI code, the corresponding message flow graph (MFG) is constructed using the approach which is described in our previous paper [22]. Next, we add the parameter dependency edges in MFG to get message flow dependency graph (MFDG). From MFG, we can get the test paths and test cases using the methodology given in [31] which are to be prioritized using our proposed prioritization approach. Our proposed approach consists following steps.

1. Compute the costs to each nodes of MFDG representing its impact using forward slicing.
2. Compute the costs to the edges of MFDG as per the criticality of message.
3. Calculate the costs of the paths generated from MFG.
4. Prioritize the order of test cases to be generated from test paths.

Computing the costs to the nodes of MFDG
In our prioritization technique, we apply forward slice to provide the weight to each of the nodes of MFDG. This weight represents the number of nodes affected by making modification at that node. Node weight of a node (Ni) in MFDG is denoted as the number nodes affected by Ni in the MFDG. Forward Slice (NFS) of a node of MDG is applied to calculate this. The calculation of weight of node Ni using forward Slice FS of node Ni is given in Equ.2.

$$\text{Weight (Ni)} = \text{FS (Ni)} \quad (2)$$

Algorithm for forward slicing: In ForwardSlice algorithm, mark the node as visited by traversing each edge at most once. than. Initially no nodes are marked as visited. Whenever a node is passed as an argument to ForwardSlice() checked the received node whether it is marked or not previously. ForwardSlice() marked the node if it is not visited previously. Then it traverses all the outgoing edges from the marked node. The function is executed recursively. The function ForwardSlice is terminated if the node is marked. The total number of affected nodes found for the node Ni is computed using ForwardSlice algorithm (i.e. FS (Ni)) will be considered as effect of that node Ni (i.e Weight(Ni)) in a particular scenario.

Algorithm1
Algorithm: **ForwardSlice**
Input: A MFDG
Output: Forward slice of each node

Initialize $S_j = \Phi$ and $NS_j = 0$; // Sj represents slice and NS_j represents visiting status of j-th node
Call FowardSlice (N);
ForwardSlice ($Node_i$)

```
begin
    if NSj=1
            return (0);
       else
            begin
            NSj =1   /* Nodei is marked as visited */
            Search for Ψj = { Nodej | Nodej are the dependency node on Nodei }
            Set Sj = Sj ∪ Ψj
            for (each node Nodej Є Ψj )
            ForwardSlice (Nodej);        /*Recursively called*/
            end
 end
```

Computing the costs edges to nodes of MFDG
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 ∈ E connecting two consecutive nodes Ni and Ni+1of MFG is computed using Equ. 3.

$$Weight(e_i)=FAN\ IN(N_i)\times FAN\ OUT(N_{j+1}) \quad (3)$$

Where FAN IN(N_i) is the number of incoming edges of node N_i
and FAN OUT (N_{i+1}) is the number of outgoing edges of node N_{i+1}.

To determine the Weight of an edge in an MFG, depth first search (DFS) traversal algorithm is used starting from start node of the MFG. When traversing the MFG, loops are executed at most once to stay away from path explosion [23,24]. We determine the incoming and outgoing control flow edges using information flow model. Then, we compute the Weight of each edge of the MFG. Each path from the root node to the end node of MFG corresponds to a scenario of the use case.

Calculating the costs of the message sequence paths
Message sequence Path weight: Message Sequence Path of MFG is a basic path MP_k ={N_1,--e_1--N_2--e_2--N_3, . . .,e_m---Nm_{+1}}, where e_1, e_2, . . .,e_m are edges and N_1,N_2,…,N_{m+1} are nodes on path MP_k in an MSG,. The Path Weight of MP_k is represented as PW (P_k) and is defined as

$$PW(P_k)=\sum_{i=1}^{m+1} Weight(Ni)+\sum_{i=1}^{m} Weight(ei) \quad (4)$$

Given the message flow graph (MFG), the work of prioritization technique is to determine the values of $_{cri}$Weight for each edge and $_{imp}$Weight for every node, and $_{psc}$Weight for each path in the TFG.

Prioritize the test cases : Message sequence path weight is used to prioritized the corresponding test cases. Equ. 4 is used to calculate message path weight. Let TP_i is a message sequence path in a MFG and tc_i is a test case subsequent to message sequence path TP_i. For message sequence path TP_i the values of Weights (TP_i) are computed. Then the computed value will be assigned to the subsequent test case. Then, the prioritizations of test cases are to be made on the decreasing order of message sequence path weight value.

V. Case study

We have taken a case study of Issue book use case of Library information system to explain the working of our proposed method for prioritizing the test cases generated from sequence diagram. The UML Sequence Diagram model is designed using StarUML which is shown in Figure 2. First, the user login to the system. If the user is a registered member having correct user-id and password then the book name will be entered by the user. Otherwise, it will display error message. Then the book is searched by the system. Now if the searched book is available and the user has not exceed the maximum number of books he/she is entitled for and still book is not issued to the user then consequently an error message will be displayed. If the book is issued to the user, the book status will be updated. After the transaction is over, system will be logged out.

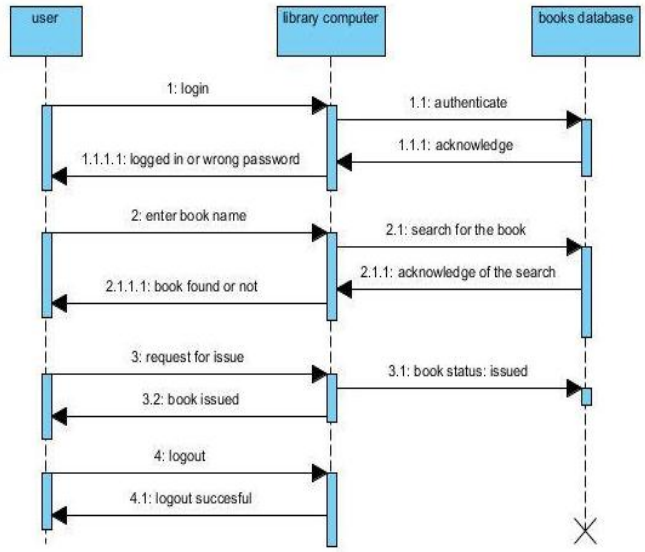
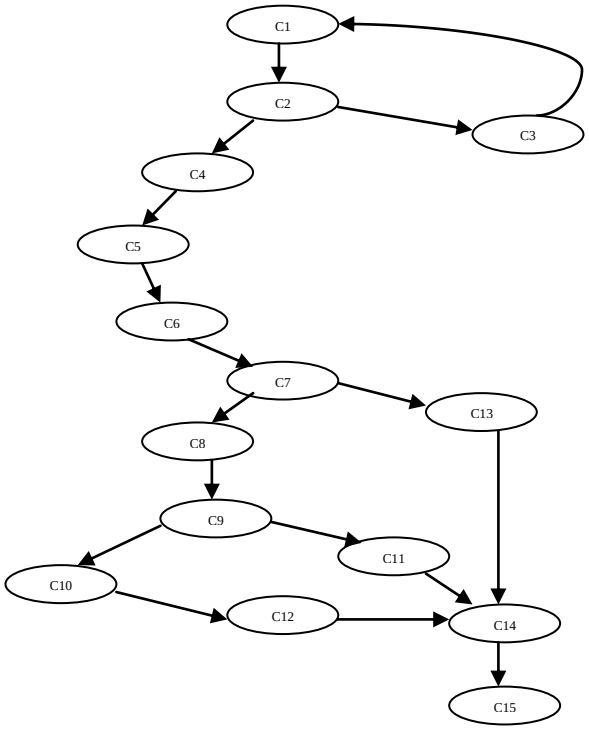


Figure 2. Sequence diagram for issue book use case of Library Information System



Label	Message
C1	Login
C2	Authentication
C3	Wrong password
C4	Enter book name
C5	Book search
C6	Acknowledgment
C7	Book search result
C8	Request issue
C9	Status checking
C10	Book issued
C11	Acknowledge(limit exceed)
C12	Book status updated
C13	Book not found
C14	Log out
C15	Log out successfully

Figure 3. Message Flow Graph (MFG) with node label for Issue Book

Fig. 3 shows the MFG of the sequence diagram given in Fig 2. which is designed using control flow among the messages of sequence diagram.

Possible Test Paths generated from MFG shown in Fig. 3, using our approach [22] are shown below

TP1: C1-C2-C4-C5-C6-C7-C13-C14-C15
TP2: C1-C2-C3-C1-C2-C4-C5-C6-C7-C13-C14-C15
TP3: C1-C2-C4-C5-C6-C7-C8-C9-C11-C14-C15
TP4: C1-C2-C3-C1-C2-C4-C5-C6-C7-C8-C9-C11-C14-C15
TP5: C1-C2-C4-C5-C6-C7-C8-C9-C10-C12-C14-C15
TP6: C1-C2-C3-C1-C2-C4-C5-C6-C7-C8-C9-C10-C12-C14-C15

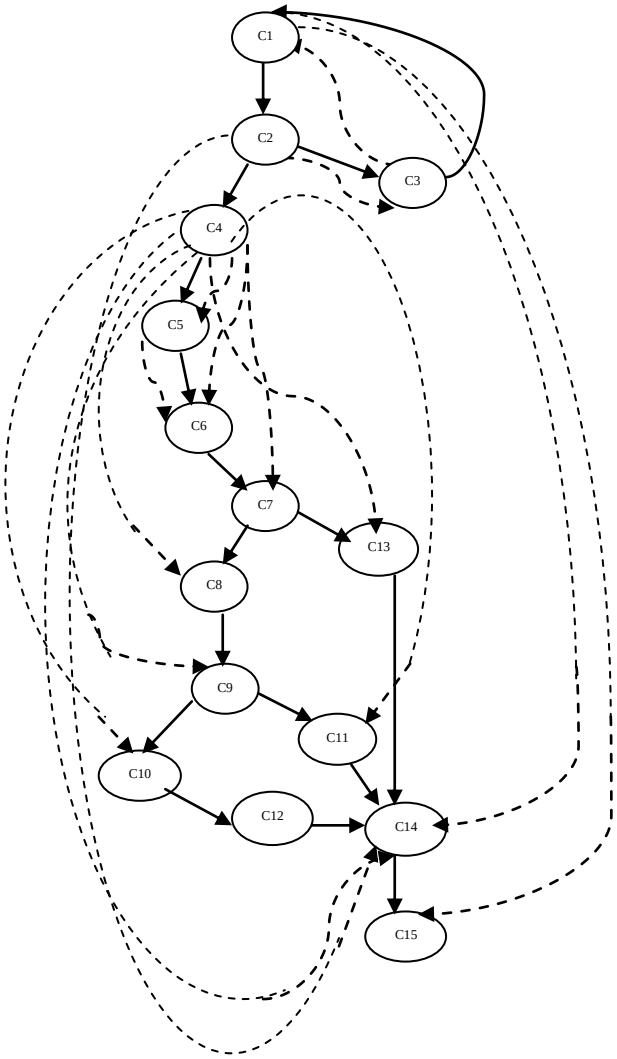


Figure 4. Message Flow Dependency Graph (MFDG) for Issue Book

Table 1 (a) Node Weights of MFDG

Node	Nodes affected	Node weight
S1	C1, C14,C15	3
S2	C2,C3,C14	3
S3	C3,C1	2
S4	C4,C5,C6,C7,C10,C12,C13	7
S5	C5,C6,C8	3
S6	C6,C7,C8	3
S7	C7,C8,C13	3
S8	C8	1
S9	C9,C10,C11,C12	4
S10	C10,C12	2

S11	C11	1
S12	C12	1
S13	C13	1
S14	C14,C15	2
S15	C15	1

Table 1 (b) Edge Weights of MFDG

Edge	weight
C1-C2(e1)	2
C2-C3(e2)	1
C2-C4(e3)	1
C3-C1(e4)	1
C4-C5(e5)	1
C5-C6(e6)	1
C6-C7(e7)	2
C7-C8(e8)	1
C7-C13(e9)	1
C8-C9(e10)	2
C9-C10(e11)	1
C9-C11(e12)	1
C10-C12(e13)	1
C11-C14(e14)	1
C12-C14(e15)	1
C13-C14(e16)	1
C14-C15(e17)	3

A. Test case Prioritization and Analysis

We compute the forward slice of each node using the algorithm "Forward Slice" given in the Algorithm 1.Using Equ.2 the node weights are calculated. The edge weights are calculated using information flow model given in Equ.3. Lastly, the overall weight is calculated for each test sequence path using the proposed metric presented in Equ.4. The calculated weights for Fig.4 are presented in Table 2.

We can prioritize the test cases of corresponding message paths in order of decreasing weights. Hence, the prioritized test case order is either tc6, tc4, tc2, tc5, tc3, tc1 to detect as faults earlier.

Table 2 Basic path sequence -wise weight calculation

Path	Corresponding test cases	Path Sequence	Node Weight	Edge Weight	Weight	Priority order
TP 1	tc_1	C1-C2-C4-C5-C6-C7-C13-C14-C15	26	12	38	VI
TP 2	tc_2	C1-C2-C3-C1-C2-C4-C5-C6-C7-C13-C14-C15	34	16	50	III
TP 3	tc_3	C1-C2-C4-C5-C6-C7-C8-C9-C11-C14-C15	31	15	46	V
TP 4	tc_4	C1-C2-C3-C1-C2-C4-C5-C6-C7-C8-C9-C11-C14-C15	39	19	58	II
TP 5	tc_5	C1-C2-C4-C5-C6-C7-C8-C9-C10-C12-C14-C15	33	16	49	IV
TP 6	tc_6	C1-C2-C3-C1-C2-C4-C5-C6-C7-C8-C9-C10-C12-C14-C15	41	20	61	I

B. Complexity Analysis

Let n be the number of nodes in the MFDG for representing the model. So the number of edges is n-1. Thus the worst case time complexity to calculate the cost of node can be $O(n^2)$ (i.e. $n \times (n-1)$). For the computation of weight of edge, any edge of MFDG is visited at most once. So if E is the total number of edges, then the time complexity will be O(E).

C. Efficiency Measures

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 APFD is calculated using Equ.1 and represented by values from 0 to 100. The rate of fault detection is faster if the value of APFD is higher.

Table 3 Fault detection using Non-Prioritized Test cases

Test Cases	Faults												
	FS_1	FS_2	FS_3	FS_4	FS_5	FS_6	FS_7	FS_8	FS_9	FS_{10}	FS_{11}	FS_{12}	Total
tc_1	*	*					*	*					4
tc_2	*	*	*				*	*				*	6
tc_3	*			*	*			*		*	*		6
tc_4	*		*	*	*			*		*	*	*	8

tc_5	*				*	*		*	*	*			6
tc_6	*		*		*	*		*	*	*		*	8

Table 4 Fault detection using Prioritized Test cases

Test Cases	Faults												
	FS_1	FS_2	FS_3	FS_4	FS_5	FS_6	FS_7	FS_8	FS_9	FS_{10}	FS_{11}	FS_{12}	Total
tc_6	*		*		*	*		*	*	*		*	8
tc_4	*		*	*	*			*		*	*	*	8
tc_2	*	*	*				*	*				*	6
tc_5	*				*	*		*	*	*			6
tc_3	*			*	*			*		*	*		6
tc_1	*	*					*	*					4

Table 3 shows fault detection by using Non-Prioritized test cases in test suite TS that detects fault FS_i, for issue book of library information system given in Fig.1.There are twelve numbers of faults taken for this given example and six numbers of test cases are selected for fault detection.

Here, we have considered p=12 and q=6. Now, APFD value for a non-prioritized test suite TS (i.e. tc1, tc2, tc3, tc4, tc5, tc6) is

$$APFD = 1 - \frac{1+1+2+3+3+5+1+1+5+3+3+2}{6*12} + \frac{1}{2*6} = 0.66$$

Using our methodology, the order of prioritized test cases TS′ can be found as : tc_6, tc_4, tc_2, tc_5, tc_3, tc_1. For p = 12 and q = 6 shown in Table 3 we found the following APFD value from Table 4. for the prioritized test cases are as follows.

$$APFD = 1 - \frac{1+3+1+2+1+1+3+1+1+1+2+1}{6*12} + \frac{1}{2*6} = 0.83$$

So, after comparing the APFD values of the prioritized and non-prioritized test cases. It is observed that the APFD value of prioritized test suite of is increased by 15% than the non-prioritized test suite.

VI. Results & Comparison with related work

Test Case Prioritization (TCP) techniques plan the ordering of execution of test cases in a test suite which increases the performance of testing or retesting by increasing the rate of fault detection. The test cases of higher priority in a test suite can enhance the goal than a random ordered test suite. Here, the complexity and necessity of test paths are of major concerns. The summary of comparison of results of our proposed approach with traditional approach using APFD for different selected software is given in Table 5. From Table 5, it is observed that our model-based prioritization approach helps to increase performance of testing by percentage of fault detection for all considered software as compared to the traditional approach [7]. The graph representation of Fault detection percentage using Traditional vrs proposed approach is shown in Fig. 5. Hence, our proposed approach is generates effective prioritized test suites.

Various Test Case Prioritization approaches describes in the literature [17,28,7,5] select regression based test cases by analyzing the source code. Other approaches [5,21,25,26] considered only data or control dependencies of program parameters. They [17] considered the test case dependencies for test case prioritization. We have proposed a UML based prioritization of test case approach using forward slice and information flow model in addition to dependencies. Further, it was presume by some existing methods that each one of the test cases are independent.. The forward slice helps to find elements having some dependencies on parameters of message. We have also considered the complexity and criticality of the test path for TCP purpose.

Kundu et al. [16] reported the overlapping scenario sub path from other scenario path to get the analytical solution to be used in project management activities. They showed efficiency of their approach by an experiment. In comparison to their approach, we have used data and control dependency

Table 5 Summary of comparison between Traditional vrs proposed approach

Experimental Case Studies	TTC	TFI	Fault detection % for using traditional method	Fault detection % using our proposed approach	Fault detection % increased using APFD
Online Hotel Management System	45	39	63.81	76.45	12.64
Vending Machine System	36	42	71.56	82.32	10.76

University Library Information System	63	58	69.40	81.27	11.87
Online Bus ticket Booking System	59	62	65.76	81.89	16.13
Online Biometric System	48	45	66.58	79.91	13.33
Organisational workflow automatic System	37	28	71.14	82.38	11.24

TTC: Total number of Test Cases taken
TFI: Total number of faults identified

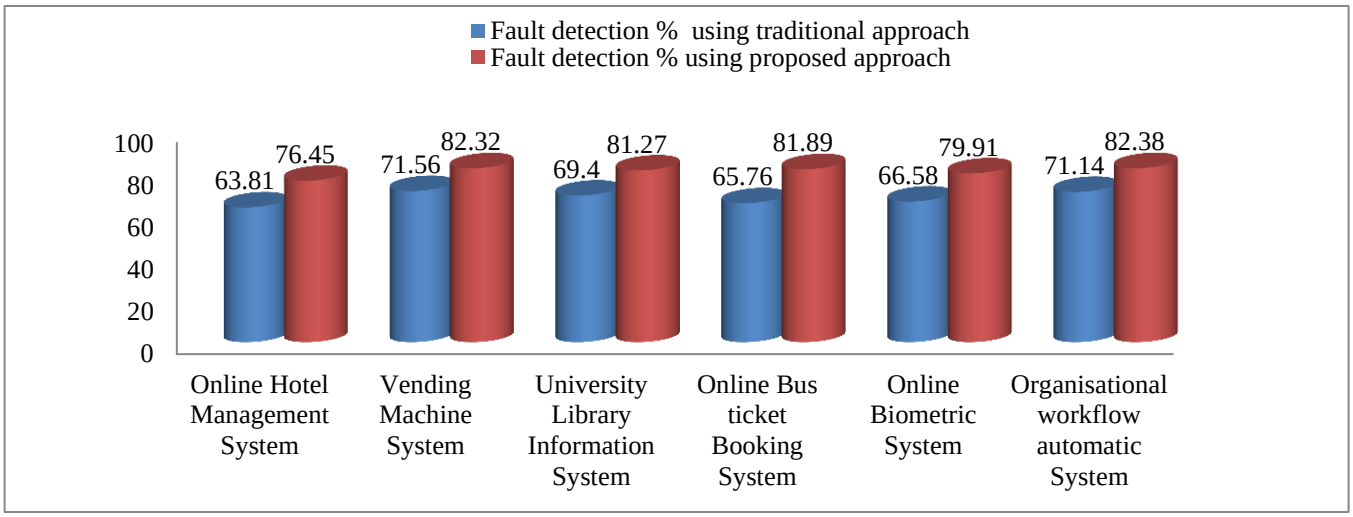


Figure5. Fault detection percentage using Traditional vrs Proposed approach

and forward slice to calculate the complexity and criticality of the path. For specific set of test cases, our approach also detects the redundant test cases.

Further some approaches [25,5], where coverage criterion based prioritization is used by identifying untested affected statements. We have observed in our case study, nearly 12 percent more regression fault detection ability as compared to Rothermel's [7] approach, which is shown in Table 5.

VII. Conclusion and Future Work

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 sequence diagrams. Our proposed approach is essentially suitable for cluster level testing. The proposed model based TCP approach is systematic, logical and easy to implement. The APFD metric is easy to compute for knowing the performance of testing. With the proposed approach, some performance goals are achieved which includes faster rate of coverage of code, higher rate of detecting faults, and faster rate in increasing the confidence in reliability of the system.

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 [7,19,18]. 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]. R. Mall, "*Fundamentals of Software Engineering*", Prentice-Hall, Springer-Verlag GmbH, 3rd Edition 2009.
[2]. 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.
[3]. H. Srikanth, L. Williams. *"On the economics of requirements-based test case prioritization"*, In Proceedings of the Seventh International Workshop on Economics-Driven Software Engineering Research, 2005.
[4]. C. E.Williams, "*Software testing and the UML"* In Proceedings of the International Symposium on Software Reliability Engineering, (ISSRE '99), Boca Raton, FL, November 1999.

[5]. S. Elbaum, A.G.Malishevsky, G.Rothermel, *"Test case prioritization: A family of empirical studies"*, IEEE Transactions on Software Engineering, Vol. 28,Issue.2, pp. 159-182, 2002.

[6]. 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.

[7]. G. Rothermel, R. H.Untch, C.Chu, M. J.Harrold, "*Prioritizing test cases for regression testing"*, Software Engineering, Vol. 27,Issue.10, pp. 929-948, 2001.

[8]. M.Weiser, *"Program slicing",* In Proceedings of the 5th International Conference on Software, San Diego, Calif, USA, pp. 439–449, 1981.

[9]. M. Weiser, "*Programmers Use Slices when Debugging",* Communications of the ACM 25, vol.7, pp.446-452, 1982.

[10]. A. Besz´edes, *"Global dynamic slicing for the C language,"* Acta Polytechnica Hungarica, vol. 12, no. 1, pp. 117–136, 2015.

[11]. X. B. Li, C. X. Fan, J. Pang, and J. J. Zhao, *"A model for slicing JAVA programs hierarchically,"* Journal of Computer Science and Technology, vol. 19, no. 6, pp. 848–858, 2004.

[12]. H. W. Alomari, M. L. Collard, J. I. Maletic, N. Alhindawi, and O. Meqdadi, *"srcSlice: very efficient and scalable forward static slicing,"* Journal of Software: Evolution and Process, vol. 26, no.11, pp. 931–961, 2014.

[13]. J. Silva, *"A vocabulary of program slicing-based techniques"*, ACM Computing Surveys, vol. 44, no. 3, article 12, 2012.

[14]. Z. Awedikian, K. Ayari, G . Antoniol, *"MC/DC Automatic Test Input Data Generation"*, In Proceedings of the 11th Annual conference on Genetic and evolutionary computation, ACM, pp. 1657-1664, 2009.

[15]. M. Weiser, *"Program slicing", IEEE Transactions on Software Engineering,* Vol. 10, No. 4,pp.352–357, 1984.

[16]. D. Kundu, D.Samanta, *"A novel approach of prioritizing use case scenarios"*, Asia–Pacific Software Engineering Conference (APSEC), IEEE Computer Society, Washington, DC, pp. 542-549, 2007.

[17]. C. R.Panigrahi, R. Mall, *"Model-based regression test case prioritization"*, ACM SIGSOFT Software Engineering Notes, Vol. 35,No.6, pp. 1-7, November 2010.

[18]. P.G. Sapna , A. Balakrishnan*"An Approach for Generating Minimal Test Cases for Regression Testing"*, Procedia Computer Science, Vol. 47,pp. 188 – 196, 2015

[19]. 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.

[20]. S. Gupta , H. Raperia , E. Kapur , H. Singh, A. Kumar,"A Novel Approach for test case Prioritization", International Journal of Computer Science, Engineering and Applications (IJCSEA) Vol.2, No.3, 2012.

[21]. D. Jeffrey, N.Gupta, *"Test case prioritization using relevant slices"*, International Computer Software and Applications Conference (COMPSAC). IEEE Computer Society, Washington, DC, pp. 411–420, 2006.

[22]. 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.

[23]. C. Mingsong, Q. Xiaokang, L. Xuandong, *"Automatic test case generation for UML activity diagrams",* In International workshop on Automation of software test, pp. 2-8, 2006.

[24]. D. Kundu, D. Samanta,*" A Novel Approach to Generate Test Cases from UML Activity Diagrams",* Journal of Object Technology, Vol. 8, No. 3, pp.65-83, 2009.

[25]. G.Rothermel., R. H.Untch, C.Chu, M. J.Harrold, *"Prioritizing test cases for regression testing"*, Software Engineering, Vol. 27,Issue.10, pp. 929-948, 2001.

[26]. S. Elbaum, G. Rothermel, S. Kanduri, and A. Malishevsky, "Selecting a cost-effective test case prioritization technique", Software Quality Control, Vol.12, Issue.3, pp.185–210, 2004.

Authors Profile

Santi Swarup Basa received his MCA degree from BPUT,Rourkela, M.Tech (Comp.Sc.) from F.M.University, Balasore and pursuing his Ph.D at North Orissa University, Baripada He is currently working as Assistant Professor in Department of Computer Science, North Orissa University, Baripada. His research interests include software engineering and Soft Computing. He has published few research papers in different International /National Journals and conferences.

Santosh Kumar Swain received his Ph.D. from KIIT University,India. His academic interests lie in Software Engineering, Web Engineering, Human-Computer Interaction, Cloud Computing, Data Mining and Wireless Sensor Network. He has published research papers in 25 international journals and 12 international conferences. He is also the co-authored of three books in the area of computer science. He is the member of LMISTE, MCSI.

Durga Prasad Mohapatra received his Ph.D. from Indian Institute of Technology Kharagpur and M. E. from Regional Engineering College (now NIT), Rourkela. He joined the faculty of the Department of Computer Science and Engineering at the National Institute of Technology, Rourkela in 1996, where he is now Professor. His research interests include software engineering, real-time systems, discrete mathematics and distributed computing. He has published more than hundred research papers in these fields in various international Journals and conferences. Dr. Mohapatra has been teaching software engineering and discrete mathematics to UG and PG students at NIT Rourkela for the past twenty years. He has received Young Scientist Award for the year 2006 by Orissa Bigyan Academy. He has also received Prof. K. Arumugam National Award and Maharashtra State National Award for outstanding research work in Software Engineering for the years 2009 and 2010 respectively by Indian Society for Technical Education (ISTE), NewDelhi, Bharat Sikshya Ratan Award by Global Society for Health and Educational Growth, Delhi for the year 2011. He has received three research projects amounting Rs. 36 Lakhs from DST and UGC, Govt. of India. Currently, he is a senior member of IEEE. Dr. Mohapatra has co-authored the book *Elements of Discrete Mathematics: A computer Oriented Approach* published by Mc-Graw Hill Education.

