Model Based Test Case Prioritization for Testing Component Dependency in CBSD Using UML Sequence Diagram

Arup Abhinna Acharya
School of Computer Engineering
KIIT University Bhubaneswar, India
aacharyafcs@kiit.ac.in

Durga Prasad Mohapatra
Department of Computer Science & Engineering
National Institute of Technology
Rourkela, India
durga@nitrkl.ac.in

Namita Panda
School of Computer Engineering
KIIT University
Bhubaneswar, India
npandafcs@kiit.ac.in

***Abstract*—Software maintenance is an important and costly activity of the software development lifecycle. To ensure proper maintenance the software undergoes regression testing. It is very inefficient to re execute every test case in regression testing for small changes. Hence test case prioritization is a technique to schedule the test case in an order that maximizes some objective function. A variety of objective functions are applicable, one such function involves rate of fault detection - a measure of how quickly faults are detected within the testing process. Early fault detection can provide a faster feedback generating a scope for debuggers to carry out their task at an early stage. In this paper we propose a method to prioritize the test cases for testing component dependency in a Component Based Software Development (CBSD) environment using Greedy Approach. An Object Interaction Graph (OIG) is being generated from the UML sequence diagrams for interdependent components. The OIG is traversed to calculate the total number of inter component object interactions and intra component object interactions. Depending upon the number of interactions the objective function is calculated and the test cases are ordered accordingly. This technique is applied to components developed in Java for a software system and found to be very effective in early fault detection as compared to non-prioritize approach.**

Keywords- Regression Testing, Object Interaction Graph, Test Cases, CBSD

I. INTRODUCTION

Nowadays software development is quality oriented development. Quality can be ensured by very good testing techniques. So optimizing time and cost of testing process is really a challenge for test engineers. Regression testing is a kind of testing which requires maximum effort, time and cost. In fact, it might be hard to run the whole application unattended and to simulate any asynchronous input (e.g., interactive inputs) the application may receive. In such cases, regression testing can last days or weeks and can involve substantial human effort. Hence a technique like Test case prioritization has to be devised which will lead to early fault detection.

Test case prioritization aims at finding an execution order for the test cases which maximizes a given objective function. Among the others, the most important prioritization objective is probably discovering faults as early as possible that is, maximizing the rate of fault detection. In fact, early feedback about faults allows anticipating the costly activities of debugging and corrective maintenance, with a related economical return. When the time necessary to execute all test cases is long, prioritizing them so as to discover most faults early might save substantial time, since bug fixing can start earlier.

The major challenges in CBSD are testing component dependency. CBSD uses the reusable components as the building blocks for constructing the complex software system (component based system). Component based system promotes the software quality and productive. This building block approach has been increasingly adopted for software development, especially for large-scale software systems. A component based software often consists of a set of self contained and loosely coupled components allowing plugand- play. The components may be implemented by using different programming languages, executed in various operational platforms distributed across geographic distances; some components may be developed in-house, while others may be the third party off-the-shelf components of which the source code may not be available to the developers. So the cost of maintaining the component based software is comparatively more than the maintenance of conventional software system. So when we want to modify or add a component and apply the regression testing, it incurs more cost and time. So to reduce these two factors we use a test prioritization technique which is based on a criterion like maximum interactions between the components performed due to a test case during component interaction. The test case having maximum interactions given higher priority and executed first so that the debugger

will not sit idle as a result fault will be detected early. In this paper for describing each component we have taken the help of sequence diagrams, then a Object Interaction Graph (OIG) from sequence diagrams is constructed which shows the interrelation among the components. A new test prioritization algorithm is presented which is applied on OIG to count the maximum number of inter component interactions and intra component interactions made by the test cases.

Previous work on test case prioritization [1, 2, 3, 4, 5] is based on the computation of a prioritization index, which determines the ordering of the test cases (e.g., by decreasing values of the index). For example, the coverage level achieved by each test case was used as a prioritization index [3]. Another example is a fault proneness index computed from a set of software metrics for the functions exercised by each test case [1].

P.R. Srivastava [18] suggested prioritizing test cases according to the criterion of increased APFD(Average percentage of Faults detected) value. He proposed a new algorithm which could be able to calculate the average number of faults found per minute by a test case and using this value sorts the test cases in decreasing order. He also determined the effectiveness of prioritized test case(more APFD value) compared to non-prioritized test case(less APFD value). G. Rothermel et. al. [19] have described several techniques for test case prioritization and empirically examined their relative abilities to improve how quickly faults can be detected by those suites. Here more importance is given to coverage based prioritization. The authors applied these techniques to the base version of a program rather than the modified version of a program, hence these techniques are otherwise known as"general prioritization techniques". The objective is to detect faults as early as possible so that the debugger will not sit idle. B. Korel et.al.[9] proposed a new prioritization technique to prioritize the test cases by using several model-based test case prioritization heuristics. Model-based test prioritization methods use the information about the system model and its behaviour to prioritize the test suite for system retesting. An experimental study has been conducted to investigate the effectiveness of those methods with respect to early fault detection. The results from the experiment suggest that system models may improve the effectiveness of test prioritization. The prioritization techniques so proposed are used in traditional software retesting, but in this work we try to use the prioritization techniques in component-based software retesting.

The test case prioritization methods can be categorized in to code-based testing and model based technique. In the code based test prioritization, source code of the system is used to prioritize the test cases. Most of the test prioritization methods [6, 10, 11, 12, 13, 14] are code based. In several test prioritization criteria were presented and their influence on the improvement of the rate of fault detection was investigated.

In model-based test prioritization [7,9] a system's model(s) is used to prioritize tests. System modelling is widely used to model state-based systems, e.g., real time systems. System models are used to capture some aspects of the system behaviour. One type of model-based test prioritization methods [7,8] are appropriate for modifications that involve changes in the model and then in the source code. The second type of model-based test prioritization methods [9] are appropriate for modifications that do not involve any changes in models (changes are only made in the source code). In this paper, we have used UML 2.0 for modelling to concentrate on the second type of model-based prioritization method.

Though several priotization techniques have been proposed previously, but the interdependency issues present in component composition in CBSD, while finding the prioritized test suit, has not been taken care of. Regression testing mainly involves testing the changes occurred in software due to addition of new components. During component composition in CBSD, the inter component dependency leads to lot of errors. So the authors have taken in to consideration the above criteria while generating the prioritized test suit to increase the APFD.

The rest of the paper is organized as follows: Section II describes the problem statement for prioritization along with a brief introduction to CBSD. The proposed model along with a case study and a comparative study are described in Section III and Section IV .The paper concludes in Section V. with the discussion on continuing work in this direction in Section VI. Due to space constraints, this paper does not include descriptions of a system model such as notations and their semantics. Interested readers are referred to any UML book such as [16] or UML manual published by OMG [15].

II. PROBLEM STATEMENT

In CBSD Component interface is defined as, it is the only way that a component communicates with the external environment. There are two kinds of interface: service providing and service required. When the services are provided by an interface it is called service providing interface and when the interface of a component requiring a service it is called service required interface. All components should be plug-compatible i.e a service required interface can be connected to a service providing interface. We have defined a Component as follows: Component C = (P, R) , where P=P_1, P_2, P_n is the set of providing services interface,

R = R_1, R_2, R_m is the set of required services interface. The providing and required services of a component C is denoted by C.P and C.R respectively and C.P∩C.R=∅. [17]

There are two kinds of special components, one is the component without the required services, the other is the one without the providing services for other components. According to the fact the numbers of two kinds of components can be one or more.

In the Fig.1 the required services of *C1P* C2 are the union of C1.R1 and C2.R2 with the remove of satisfied services in S. With the definition of composition the providing and required services are propagated to the interface of composed component, so the composition could be carried parallel. A Component interaction graph (OIG) is used to describe the interrelation of components. A complete component interaction graph (OIG) makes the testing quite easy. A OIG is a directed graph where OIG = (V, E), V represents a set of nodes. V = VI U VC, VI is the set of interface nodes and VC is the set of component nodes, E represents the set of directed edges.

The interface is denoted by an ellipse and a component with dashed square. The interaction among components can be gained from the OIG directly.

There are two kinds of special components, one is the component without the required services, the other is the one without the providing services for other components. According to the fact the numbers of two kinds of components can be one or more.

The OIG illustration is given in figure 1:

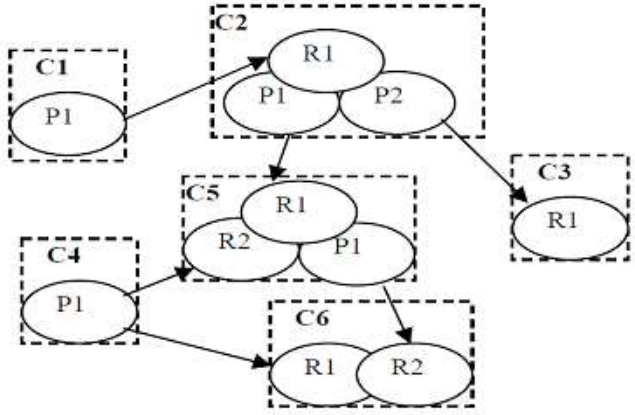

Fig. 1. Object Interaction Graph(OIG)

If there is an existing edge form C1.P1 to C2.R1 in the CIG it means the required service R1 of C2 has been satisfied by the providing service P1 of C1, which is C2.R1= C1.P1.

Practically it is not possible to perform rigorous testing. Tester has to select subset of test cases from the original test suite. This makes test case selection quite challenging. This selected regression suite should cover all the functionality i.e. adequate functional coverage and greater fault exposing potential. Due to squeezed test schedule, testing team may not able to execute all test cases from the selected regression suite. Sequencing of test cases based on some criteria helps testing team to achieve the goals whilst reducing testing cycles. Rothermel at el. [3] defines the test case prioritization problem as follows: **Given:** T, a testsuite; PT, the set of permutations of T; f, a function from PT to the real numbers.

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

Here, PT represents the set of all possible prioritizations (orderings) of T and f is a function that, applied to any such ordering, yields an award value for that ordering [3]. The objective of this research is to develop a test case prioritization technique that prioritizes test cases on the basis of detection of fault rate.

III. PROPOSED MODEL

To facilitate regression testing by optimizing the time and cost, we propose a method to prioritize the test cases by using model based prioritization method by extracting the benefits of Unified Modelling Language(UML). UML provides lifecycle support in software development and is widely used to describe analysis and design specifications of software. It is a big challenge to study the test case generation from UML diagram. In case of a Object Oriented System Design (OOSD), each component is represented by collection of objects. Due to encapsulation, the only way objects can communicate is through message passing. Whenever an event occurs, it is executed through a sequence of occurrence of message passing. We have used sequence diagram from the set of diagrams present in UML 2.0. As Sequence diagram represents various object interactions through message passing, it can act as an input to the proposed model. We are generating an Object Interaction Graph (OIG) from the sequence diagrams present. The methodology we have used for generating the graph has been discussed in Section III(A) Further in Section III(B) we have discussed how to traverse the OIG to calculate the number of inter component object interaction and intra component object interaction. Section III(B) describes about objective function evaluation and the prioritization technique.

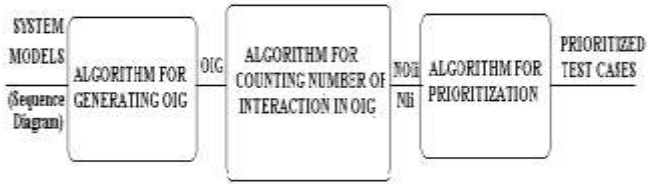

Fig. 2. A Frame Work For Generating Prioritized Test Cases

A. Generating OIG form System Models

We have used sequence diagram for system modelling. The object interactions can be very well identified using a sequence diagram. During regression testing any modification in the code will have no effect on the sequence diagram. The object interaction can be categorized into two different types. One of them is intra component object interactions and the other one is inter component object interactions. In case of intra component object interaction, the interaction between objects present within a component is considered where as in case of inter component object interaction we consider the object interactions present

between two different components. A sequence diagrams in UML are used to model how an object communicate with other objects in its life time i.e. it is used to capture the dynamic behaviour of a system. The basic elements of a sequence diagram are object s and messages [Booch, Rumbaugh and Jacobson 1998] and it shows a state machine which emphasizes the flow of control from state to state.

A Object Interaction Graph (OIG) is used to describe the interrelation of components. A complete object interaction graph (OIG) makes the testing quite easy. A OIG is a directed graph where OIG = (V, E), V represents a set of nodes. For generating Object Interaction Graph (OIG), each object present in the sequence diagram is represented as a node in the graph. The intra component object interactions form the edges of the graph and represented in BLACK color. The inter component object interactions form the edges of the graph and represented in RED color.

Algorithm: GENERATE OIG
Input: Sequence Diagrams of various components of the system representing message passing between objects
Output: Object Interaction Graph (OIG)// It is a directed graph

1. Initialize OIG to be empty
2. for i=1 to n//n is the total number of objects
3. Add a node N_i to OIG == N_i represents i^{th} node. Object shared by different components treated as a single node.
4. for i=1 to n
5. for j=1 to n
6. for each incoming message from object O_i to O_j ==All guard conditions are ignored
7. if (interaction type==intra)Establish an edge between O_i to O_j (i.e. N_i and N_j) and color it as ”BLACK” as well as append the pre and post conditions.
8. Else Establish an edge between O_i to O_j (i.e. N_i and N_j)and color it as ”RED” as well as append the pre and post conditions.
9. The possible start and end of the scenario sequences are represented with solid arrows.

B. Traversing OIG

When the OIG is generated from the system models, it has to be traversed to count the number of inter component and intra component object interactions. NOI_i represents the number of Object Interactions discovered by test case ti with in one component of the software and NI_i represents the number of Object Interactions discovered by test case ti between two different components of the software. We follow the depth first search (DFS) methodology for traversing the graph. The type of interaction is decided depending upon the color of the edge in the graph. If the edge color is found to be ”BLACK”, it represents an intra component object interaction, where as edges colored as ”RED” represents inter component object interaction

Algorithm: IN_CALCULATE
Input: Test case *ti* & Object Interaction Graph (OIG)
Output: NOI_i and NI_i

1. Initialize both NOI_i and NI_i to 0.
2. Traverse each interaction in the OIG for *ti* in DFS
3. if (edge color ==’BLACK’ && current edge is not visited already)
4. *NOIi* + + ==Increment the value for intra component interaction
5. Else
6. NI_i + + == Increment the value for inter component interaction
7. Return NOI_i and NI_i

C. Generating Prioritized Test Cases

Once we get the value for NIi and NOIi by using the algorithm described in Section III(B), prioritization process starts. For each test case ti, the value of NIi and NOIi are added. We have considered the total number of intra component interaction where as the total number of inter component object interactions is found out by multiplying it with RP i.e. total number of providing service interface and required service interface . If the faults due to component integration are detected early, it will give a better coverage.

The added result is divided with unit time U to determine value of the objective function i.e. factor criteria FCi. We try to maximize the objective function using a Greedy approach.

Algorithm: TEST_PRI
Input: Regression Test Suite T
Output: Prioritize Test Suite T’

1. Traverse the test suite T, for each test case *ti* present, call **IN_CALCULATE** (t_i) to calculate NOI_i and NI_i
2. Define some unit time U
3. Calculate objective function (FC_i) for test case *ti* as $FC_i = (NOI_i + \mathbf{RP} * NI_i)/U$. *(1)*
   ***// RP* represents total number of providing service interface**.
4. Generate T’ by Sorting the test suit T in ascending order of FC_i for each t_i .
5. Store T’ in the test case repository for regression testing.

IV. Case Study- A Cellular Network Manager

We have taken the case study of a Cellular Network Manager to explain the proposed model. We have taken into consideration two components i.e.”Dialing a Phone” and ”Cellular Network Connection”. From the sequence diagram of both the components given in Fig.3 and Fig.4, corresponding OIG are designed as given in Fig.5.

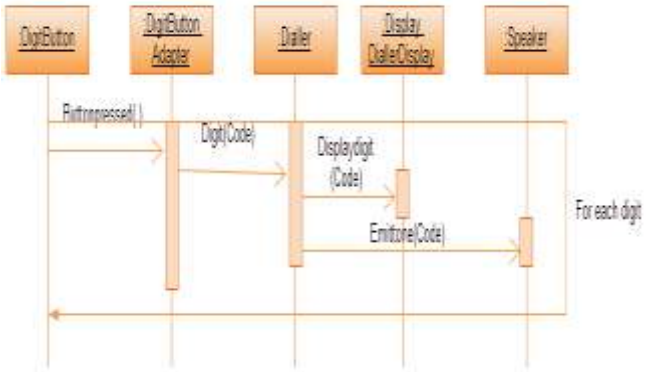

Fig. 3. Sequence Diagram for dialing the number

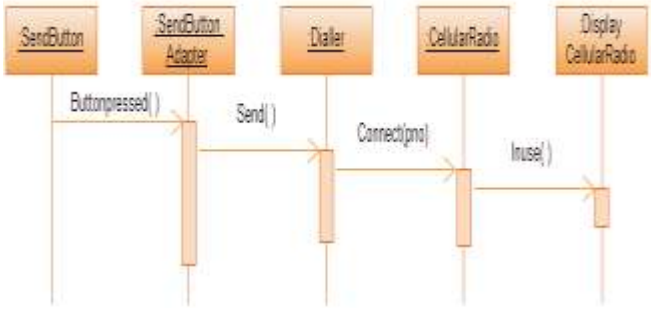

Fig. 4. Sequence Diagram for cellular phone connection

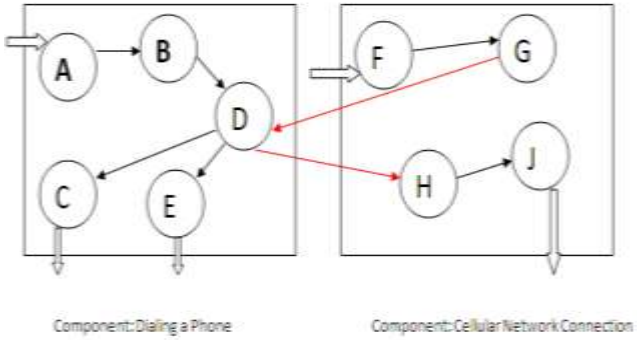

Fig. 5. OIG for a Cellular Network Manager

A: Digit Button
B: Digit Button Adapter
C: Dialer Display
D: Dialer **(Both P&R)**
E: Speaker
F: Send Button
G: Send Button Adapter **(P)**
H: Cellular Radio**(R)**
J: Cellular Radio Display

Three test cases are considered to test the prioritization algorithm. The test cases are designed to test the Dialer Display(t_1), to test the Speaker(t_2) and to test the Cellular Radio Display(t_3). The following table contains the value of NOI_i, NI_i and FC_i. Here the unit time U is considered to be 1 unit.

TABLE I: *OBJECTIVE FUNCTION (FC_i) EVALUATION*

Test Cases	NOI_i	NI_i	FC_i
t_1	3	0	3
t_2	3	0	3
t_3	2	4	6

From the table I we conclude that the prioritized test sequence is: **t_3, t_2, t_1 or t_3, t_1, t_2**

The proposed model found to be very effective as it increases the Average Percentage of Fault Detection (APFD) when it is compared with generalized model based method and few code based methods like LOC count and Function count. The comparison made is summarized in Table-II.

TABLE II
A COMPARATIVE STUDY

Name of Prioritized Technique	Approximate Increase in APFD value(%)
Code based Approach (LOC count, Function count etc.)	30
Model based Approach	35
Model Based Approach using the Dependency Criteria in CBSD	45

V. CONCLUSION

The cost and time required for regression testing can be minimized by using the prioritization technique discussed in this paper. Here we have proposed a model based prioritization method by considering the number of Object Interactions per unit time as the objective function. Here more importance is given to number of inter component object interactions present because maximum faults are expected to be present when components interact with each other. The proposed model found to be very effective as it increases the Average Percentage of Fault Detection (APFD) when it is applied to few of the projects developed in Java by java 45%-50%. This approach is mainly applicable to test the component composition in case of component based software maintenance.

VI. CONTINUING WORK

The proposed method can further be extended to prioritize test cases to perform regression testing for real time systems and distributed systems. Here the authors prioritize the test case using a model based approach. The authors are also working on adding new criterion like frequency of data base access number of state changes in UML state chart diagram etc. Two different modelling diagrams can also be integrated to find criterion to generate test cases Requirement specifications can also be used to prioritize the test cases. Test case prioritization for

concurrent systems is also a very challenging area of research due to its dynamic behaviour

REFERENCES

[1] S. Elbaum, A. Malishevsky, and G. Rothermel.*Test case prioritization: A family of empirical studies.*, IEEE Transactions on Software Engineering, 28(2):159-182, February 2002.

[2] J. M. Kim and A. A. Porter.*A history-based test prioritization technique for regression testing in resource constrained environments.*, In Proceedings of the International Conference on Software Engineering (ICSE), pages 119-129. ACM Press, May 2002.

[3] G. Rothermel, R. Untch, C. Chu, and M. J. Harrold*Test case prioritization.*, IEEE Transactions on Software Engineering, 27(10):929-948, October 2001.

[4] H. Srikanth, L. Williams, and J. Osborne.*System test case prioritization of new and regression test cases.*, In Proceedings of the 4th International Symposium on Empirical Software Engineering (ISESE), pages 62-71. IEEE Computer Society, November 2005.

[5] A. Srivastava and J. Thiagarajan.*Effectively prioritizing tests in development environment*, In Proceedings of the International Symposium on Software Testing and Analysis (ISSTA), pages 97-106. ACM Press, July 2002.

[6] J. Kim, A. Porter, *"A History-Based Test Prioritization Technique for Regression Testing in Resource Constraint Environments,"*, Proc. 24th International Conference on Software Engineering, pp. 119-129, 2002.

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

[8] B. Korel, G. Koutsogiannakis, L. Tahat,*"Model-Based Test Prioritization Heuristic Methods and Their Evaluation"*, 3rd ACM Workshop on Advances in Model Based Testing, A-MOST, 2007.

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

[10] Z. Li, M. Harman, R. Hierons,*"Search Algorithms for Regression Test Case Prioritization,"*,IEEE Transactions on Software Engineering, vol. 33, No. 4, pp. 225-237, 2007.

[11] G. Rothermel, R. Untch, C. Chu, M. Harrold,*"Test Case Prioritization: An Empirical Study,"*,Proc. IEEE International Conference on Software Maintenance, pp. 179-188, 1999.

[12] G. Rothermel, R. Untch, M. Harrold,*"Prioritizing Test Cases For Regression Testing,"*,IEEE Transactions on Software Engineering, vol. 27, No. 10, pp. 929-948, 2001.

[13] A. Srivastava, J. Thiagarajan,*"Effectively Prioritizing Tests in Development Environment,"*,Proc. ACM International Symposium on Software Testing and Analysis, ISSTA-02, pp. 97- 106, 2002.

[14] W. Wong, J. Horgan, S. London, H. Agrawal,*"A Study of Effective Regression Testing in Practice,"*,Proc. International Symposium on Software Reliability Eng., pp. 230-238, 1997.

[15] UML 2.0 Reference Manual, Object Management Group, 2003.

[16] Schneider, G., and winters, J.P., Applying Use Cases, Second edition, Addison Wesley,2001.

[17] Arup Abhinna Acharya, Sisir Kumar Jena, *" Component Interaction Graph: A new approach to test component composition"*,Journal of Computer Science and Engineering, Volume 1, Issue 1, May 2010.

[18] P. R. Srivastava, *"Test Case Prioritization"*, Journal of Theoritical And Applied Information Technology 2008 JATIT

[19] G. Rothermel, R. H. Untch, C. Chu ,M. J. Harrold *"Test Case*

Prioritization:An Emperical Study", in Proceedings of the 24th IEEE International Conference Software Maintenance (ICSM '1999) Oxford, U.K,September,1999 .

[20] GB. Korel, G. Koutsogiannakis, *"Experimental Comparsion of Code Based and Model model Based Test prioritization "*, IEEE 2009.

AUTHORS PROFILE

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

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

Namita Panda is an Assistant Professor in the School of Computer Engineering, KIIT University, Bhubaneswar, Odisha, INDIA. She received her Master's degree from KIIT University Bhubaneswar. Her research areas include Object Oriented Software Testing, Parallel Processing and Computer Architecture. She has published papers in national and international level proceedings. She is having seven years of teaching experience. She is a member of ISTE. She can be reached at npandafcs@kiit.ac.in.