Proc. of the Intl. Conf. on Advances in Computer Science and Electronics Engineering -- CSEE 2014

ISBN: 978-1-63248-000-2 doi: 10.15224/ 978-1-63248-000-2-45

Regression Test Framework based on Extended System Dependence Graph for Object-Oriented Programs

Samaila Musa
Abu Bakar M.D. Sultan
Azim Abd Ghani
Dr. Salmi Bahrom

Abstract - This paper presents a regression testing framework for object-oriented software based on extended system dependence graph model of the affected program. The approach is based on semantic analysis of the code. The goal is to identify changes in a method's body due to data dependence, control dependence and dependent due to object relation such as inheritance and polymorphism. To find the affected statements due to changes in the program, we used affected statement as slicing criterion to performed slicing on the constructed graph. The methods affected are determined by analysis of the ESDG based on the affected statements. Test cases that execute the affected methods are selected from an existing test suite. New test cases are generated when necessary. The selected test cases are prioritized by assigning weight to the affected methods. A case study will be reported to provide evidence of the feasibility of the approach and its benefits in increasing the rate of fault detection and reduction in regression testing effort.

Keywords-regression testing, regression test framework, regression test case prioritization, system dependence graph.

I. Introduction

Software maintenance activity is an expensive phase account for nearly 60% of the total cost of the software production [1]. Regression testing is an important phase in software maintenance activity to ensure that modifications due to debugging or improvement do not affect the existing functionalities and the initial requirement of the design [2] and it almost takes 80% of the overall testing budget and up to 50% of the cost of software maintenance [3].

Samaila Musa
Abu Bakar M.D. Sultan
Azim Abd Ghani
Salmi Bahrom
Faculty of Computer Sci. & Info. Technology, University Putra Malaysia
Malaysia

Regression test selection is a way that test cases are selected from an existing test suite, that need to be rerun to ensure that modified parts behave as intended and the modification have not introduce sudden faults. Reduction in the number of test cases to be used in testing modified program means reduction in the cost associated with regression testing.

Identifying test cases that exercised modified parts of the software is the main objective of regression test selection. The challenge in regression testing is identifying and selecting of best test cases from the existing test suite, and selecting good test cases will reduce execution time and maximize the coverage of fault detection.

Regression testing approach can be based on source code, i.e., code-based and based on design, i.e., design-based, many of them were proposed by the researchers. The more safe and easy to make are the approaches that generate the model directly from the source code of the software.

Researchers have proposed many code-based approaches [3, 4, 5, 6, 7] by identifying modifications in the level of source code, but the authors focus on the procedural-based programming which are not suitable in object-oriented programming widely today used in software development. Other researchers [2, 8, 9, 10, 11, 12, 13,16,17] address the issues of object-oriented programming but do not consider some basic concept of object-oriented features (such as inheritance, polymorphism, etc.,) as a bases in identifying changes.

An approach was presented in [13] based on the concept of Control Call Graphs (CCG), reduced form of Control Flow Graph (CFG). This graph is a directed in which the nodes represent decision points, an instruction or a block of statements. An arc in the graph linking nodes Ni to Nj means that the statements corresponding to node Ni will be executed first, followed by the statements in node Nj. The control flow (method calls and distribution of the control flow) in the system are provided by CCG. The technique is more precise and captures the structure of calls and related control than the traditional Call Graph (CG). However, it is difficult to extracting information about changes in the source code that may not have direct impact on the method call.

A heuristic-based test case prioritization approach for object-oriented programs was presented in [17]. The

Proc. of the Intl. Conf. on Advances in Computer Science and Electronics Engineering -- CSEE 2014

ISBN: 978-1-63248-000-2 doi: 10.15224/ 978-1-63248-000-2-45

technique was based on analysis of dependence model, as improvement for technique presented in [16]. The authors constructed a dependence model of a program from its source code, and when the program is modified, the model is updated to reflect the changes. The test cases that covered the affected nodes are selected for regression testing. The selected test cases are then prioritized by assigning initial weights of 1(one) to the affected nodes. The weights of the affected nodes in the selected test cases covered by previous execution of a test case are reduced to 0.5. But, the weight 0.5 may have effect in the selection if the numbers of covered nodes are many in a test case, which may result in selection of test case that is not much relevant, which will result in increase of regression testing time.

In this paper we present an approach that will select best test cases from existing test suite T used to test the original program P by using Extended System Dependence Graph (ESDG) [15] as an intermediate to identify the changes in P, at statements level. Identification of changes using this kind of graph will leads to précised detection of changes. The changed statements will be used to identify affected methods, and test cases that execute the affected methods are selected for regression testing. The selected test cases will be prioritized based on reduction in weight of the affected methods in order to increases the rate of faults detection. This approach will reduce the cost of regression testing by reducing the number of test cases to be used in testing the modified program.

Extended System Dependence Graph (ESDG) [15] is a graph that can represents control and data dependencies, and information pertaining to various types of dependencies arising from object-relations such as association, inheritance and polymorphism. Analysis at statement levels with ESDG model helps in identifying changes at basic simple statement levels, simple method call statements, and polymorphic method calls.

The rest of this paper is organized as follows. In the next section, we provide regression testing. Section 3 describes Extended System Dependence Graph (ESDG). In section 4, we present our test selection framework. Section 5 concludes this paper.

II. Regression Testing

Regression testing is a software testing activity normally conducted after software is changed, and its helps not only to ensure that changes due to debugging or improvement do not affect the existing functionalities but also the changes do not affect the initial requirement of the design. Regression test selection is an activity that select test cases from an existing test suite, that need to be rerun to ensure that modified parts behave as intended and the modification have not introduce sudden faults.

Regression test selection technique will help in selecting a subset of test cases from the test suite. The easiest way is that, the tester simply executes all of the existing test cases to ensure that the new changes are harmless and is referred as retest-all method [9]. It is the safest technique, but it is possible only if the test suite is small in size. The test case can be selected at random to reduce the size of the test suite. But most of the test cases selected randomly can result in checking small parts of the modified software, or may not even have any relation with the modified program. Regression test selection techniques will be an alternative approach.

Selected test cases that execute both the modified portion of the program and the portions that are affected by these modifications are referred to as modification revealing test cases. Regression test selection involves the selecting and running a reduced subset of test cases from the initial test suite, in order to verify the behavior of modified software and provide confidence that part of the software affected by modifications are correct. This can results to reduction in the cost of regression testing and also software maintenance.

Problem definition:

Let P be a certified program tested with test suite T, and P' be a modified program of P. During regression testing of P', T and information about the testing of P with T are available for use in testing P'.

To solve the above problem, Rothermel and Harrold [3] have outlined a typical selective retest technique that:

- Identify changes made to P by creating a mapping of the changes between P and P'
- Use the result of the above step to select a set T' subset of T that may reveal changes-related faults in P'
- Use T' to test P', to establish the correctness of P` with respect to T`
- Identify if any parts of the system have not been tested adequately and generate a new set of test case T''.
- Use T'' to test P`, to establish the correctness of P` with respect to T``
- Create T```, a new test suite and test history for P`, from T, T`, and T``.

III. Extended System Dependence Graph

In this section, we describe the dependency graph based on the approach presented in [15]. ESDG [15] was used to model object-oriented programs and is an extension of System Dependence Graph (SDG) [14] used to model procedural programs.

Extended System Dependence Graph (ESDG) [15] is a graph that can represents control and data dependencies, and information pertaining to various types of dependencies arising from object-relations such as association, inheritance and polymorphism. Analysis at statement levels with ESDG model helps in identifying changes at basic simple statement levels, simple method call statements, and polymorphic method calls.

ESDG is a directed, connected graph G = (V, E), that consist of set of V vertices and a set E of edges. A vertex v represents one of the four types of vertices, namely,

Proc. of the Intl. Conf. on Advances in Computer Science and Electronics Engineering -- CSEE 2014

ISBN: 978-1-63248-000-2 doi: 10.15224/ 978-1-63248-000-2-45

statement vertices, entry vertices, parameter, and polymorphic vertices. An edge e represent one of the six edges, namely, control dependence edges, data dependence edges, parameter dependence edges, method call edges, summary edges, and class member edges.

A. *ESDG Vertices*

- Statement vertices: Are program statements present in the methods body. Statement vertices are of two types: call vertices and simple statement vertices. Call vertices are used to represent method call statements, and all other statements such as conditionals, loops and assignment in the program are represented by simple statement vertices.

- Parameter vertices: Are used to represent parameter passing between a caller and callee method. They are of four types: formal-in, formal-out, actual-in, and actual-out. Actual –in and actual-out vertices are created for each call vertex and create formal-in and formal-out vertices for each method entry vertex.

- Entry vertices: Methods and classes have entry vertices. A class entry vertex and a method entry vertex represent an entry into a class and an entry into a method respectively.

- Polymorphic choice vertex: it is used to represent dynamic choice among the possible bindings in a polymorphic call.

B. *ESDG Vertices*

- Control dependence edge: It is used to represents control dependence relations between two statement vertices.

- Data dependence edge: It is used to represents data dependence relations between statement vertices.

- Call edge: It is used to connect a calling statement to a method entry vertex. It also connect various possible polymorphic method call vertices to a polymorphic choice vertex.

- parameter dependence edge: It is used for passing values between actual and formal parameters in a method call. It is of two types: parameter-in and parameter-out edges.

- Summary edge: It is used to represents the transitive dependence between actual-in actual-out vertices.

- Class member edge: It is used to represents the membership relation between a class and its methods. It is used to connect a class entry vertex to a method entry vertex.

Figure 1 represent the different graphical symbols used represent the different types of vertices and edges. In figure 2 and figure 3, we present a program example and its ESDG.

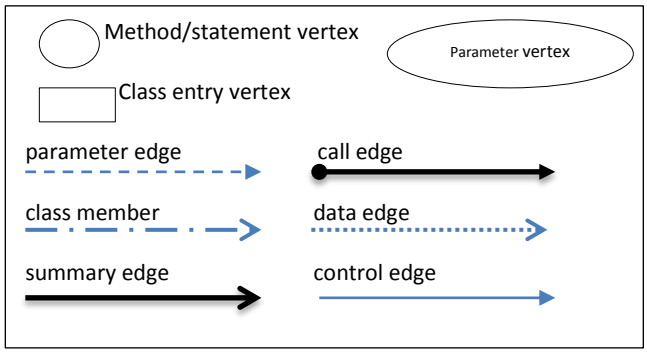

Fig. 1. Graphical symbols used to represent the different types of vertices and edges in ESDG.

```
CE1 public class Tsum {
S2   public static int i;
S3   public static int sum;
E4   public void TSum( ) {
S5            sum =0;
S6            i = 1;
   }
E7   public void calculate( ) {
S8            while (i<10) {
S9            sum = add(sum, i);
S10  i = add(i, 1);  }
```

```
S11 System.out.println("sum =
" + sum);
S12 System.out.println
("i = " + i);
   }
E13  static int add (int a, int b) {
S14    return(a+b);
   }
  }
```

Fig 2. Example of a class

IV. Regression Test Framework

This paper presents an approach for the selection of test cases T` from the test suite T to be used in testing the modified program P`. Figure 4 illustrate the various activities of the test case selection framework.

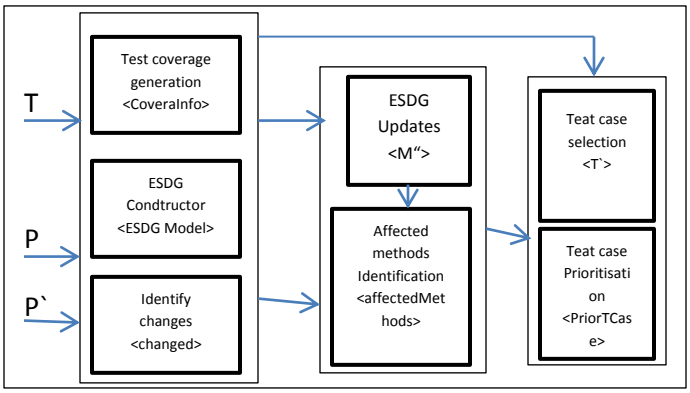

Figure 4. Framework of our approach

Proc. of the Intl. Conf. on Advances in Computer Science and Electronics Engineering -- CSEE 2014

ISBN: 978-1-63248-000-2 doi: 10.15224/ 978-1-63248-000-2-45

Our approach consists of the following phases:

A. *Identify Changes*

The changes between *P* and the modified program *P`* are identified in this step, via semantic analysis of the source code of the software. A file named changes will be used to store the identified statement level differences. This is shown in Fig. 1 by the result of identify changes phase.

The scopes of the changes in our approach are addition and deletion of object.

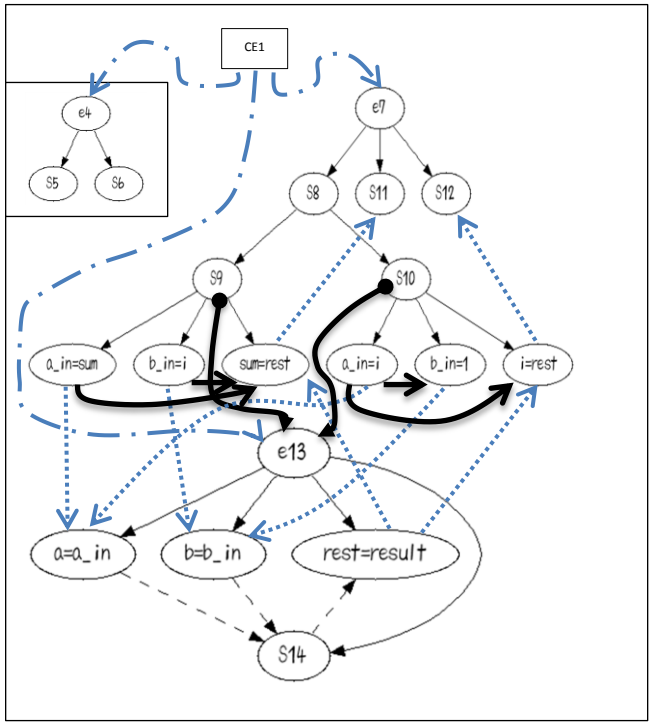

Fig 3. Partial ESDG of the program in fig 2

1) Adding object:

Adding object in ESDG can be identified by Identify changes phase. Adding of object in object-oriented programming can be addition of method call statements, or simple statements such as conditionals, loops and assignment statements in the program. Figure 5a (i, ii) and figure 5b (i, ii) represent program P and its modified version P` codes, and their corresponding ESDG of simple statements addition respectively.

Figure 6a (i, ii) and figure 6b (i, ii) represent program P and its modified version P` codes, and their ESDGs of addition of method call statement.

In fig 5a i, statements (vertices) S2, S3, S4 and S5 are control dependence on E1 (method entry vertex). Vertices S3, S4 and S5 are data dependent on S2, and S5 is data dependent on S2, S3 and S4. In fig 5a ii, statement S4 and S5 are not data dependent on S2, but are on the added statement S3a. The added statement S3a, is data dependent on S2. Statement S3a is identify as the changes between P and P`, and is saved as changed node.

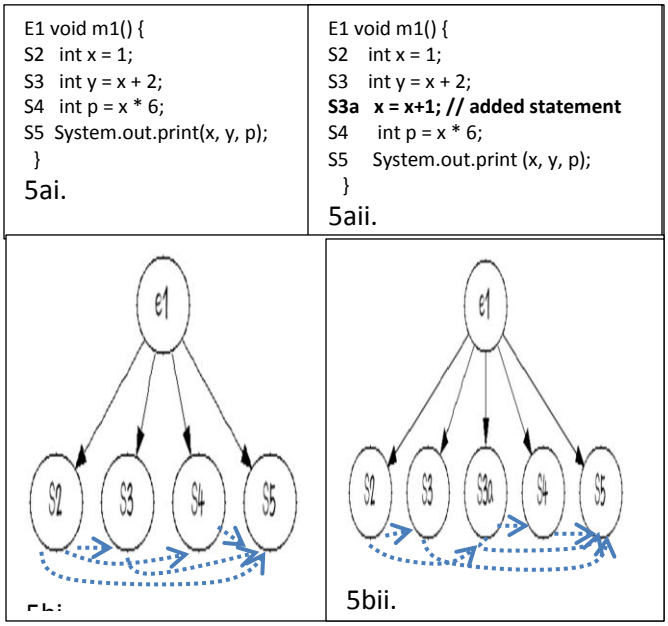

Fig 5. Program P and its modified version P`, and their ESDG of simple statement addition.

.

In fig 6aii, method call statement sum(int x, int y) was added in S6a. fig 6bii represents the method call statement added in the code in line S6a. Parameter edges were drawn from Actual_in vertices of method call to formal_in vertices of the called method. So also a parameter edge is drawn from formal_out vertex to Actual_out vertex. A simple call edge is drawn from the call statement S6a to callee method entry node, and a control dependence edge has been drawn from method entry vertex E6 to method call statement vertex (S6a). Since the output of the S6a statement is transitively dependent on the its Actual_in parameters, summary edges will be drawn from Actual_in vertices to Actual_out vertex.

```
CE1     class A {
E2          Public int x, y;
E3          void A () {
S4              x = 5;
S5              y = 7; }
```

```
E6    void increment (y) {
S6a   sum (y, 1); }// added
method call statement
E7    void sum (int x, int y) {
S8              x += y;
            }
        } // class
```

Fig 6aii. Addition of method call statement in the code

2) Deleting of object:

An example of deletion of simple statement has been presented in fig 7a, and in the case of deletion of method call statement, we will used the code and ESDG in fig 6 (aii and b). In fig 7aii, the deleted node is S4 marked by dash line. The statement vertices S5 and S6 are data dependent on S4, so before deleting S4, nodes S5 and S6 are identified by conducting forward slices on the code. Then edges from deleted node to nodes S5 and S6 are deleted and saved the identified nodes as changed nodes, and also a data

Proc. of the Intl. Conf. on Advances in Computer Science and Electronics Engineering -- CSEE 2014

ISBN: 978-1-63248-000-2 doi: 10.15224/ 978-1-63248-000-2-45

dependence edge from node S2 to the deleted node is removed due to deletion of S4.

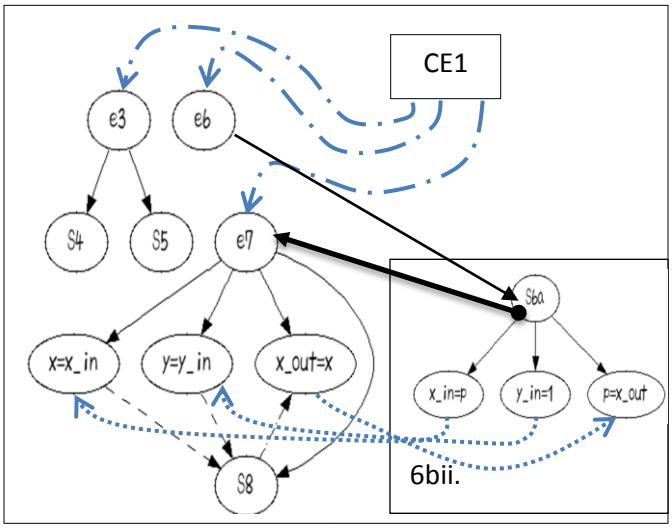

Fig 6b(i and ii). updated ESDG model for addition of method call statement

In fig 6aii, we assume the deleted method call statement is S6a. To update the model by deleting the node S6a in ESDG, first identify the changed nodes, i.e., nodes that are control dependent or data dependent or dependent due to object relation such as inheritance and polymorphism, and saved these nodes in the file named changes to be used later. Secondly, there is need to remove all the parameter edges, the simple call edge, and control dependence edge. Then the vertices are deleted.

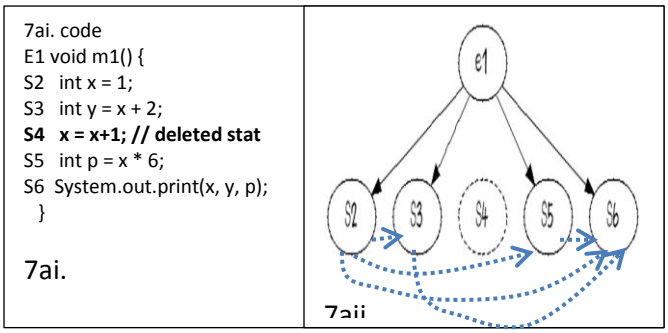

Fig 7. Deletion of a simple statement.

B. Test Coverage Generation

Program P is instrumented at the method levels. The code statements are executed with the original test suite T and to write traces for each test case in order to generate information pertaining to the specific methods that are executed for each test case. The generation of the test coverage information is perform once for a given program during one testing cycle, and the activity will not be repeated for the subsequent regression testing which will saved time. The information generated in this stage is saved in a file named coverageInfo for later use.

C. ESDG Model Constructor

ESDG model for the original program P is constructed using a technique similar to [15], and was described in section III.

D. ESDG Model Updates

The model constructed for P is updated using information from changed file during each regression testing to make it correspond to the modified program P` and the updated ESDG model is denoted by M`.

E. Affected Methods identification

To identify the affected methods, a forward slice is constructed on the updated model M` using the information from changed file. Each change node in changed file is used as slicing criterion to determine the affected nodes in each method. The affected nodes stored in changed file are used to identify the affected method. The affected methods are methods that were affected directly by the modifications in their body or as the result of control dependence or data dependence or dependent as a result of object relation such as inheritance and associations on the affected node from the updated model M`, and denoted by affectedMethod.

F. Test Case Selection

Test cases that execute the affected methods in the updated model M` are selected for regression testing, and donated as T`.

G. Test Case Prioritization

The selected test cases T` will be prioritized based on reduction in weight of the affected methods, and denoted as PriorTCase.

V. Conclusion

A test case selection framework has been proposed in our approach that selects test cases T` from test suite T to be used for rerun in regression testing. The approach used extended system dependence graph (ESDG) [15] to identify changes at statement level of source code, store the changes in a file named changed, and generate coverage information for each test case from the source code. The changed information are used to identify the affected methods, and test cases are identify that will be rerun in regression testing based on the affected methods. The selected test cases will be prioritized based on reduction in weight of the affected methods already covered by the previous execution in order to increases the rate of faults detection The technique cover the different important issues that regression testing strategies need to address: change identification, test selection, test execution and test suite maintenance.

A tool will be developed based on our proposed framework to be used in object-oriented programs, and we will compare results from our developed tool to measure the preciseness, inclusiveness and rate of faults detection.

ISBN: 978-1-63248-000-2 doi: 10.15224/ 978-1-63248-000-2-45

References

[1] S. P. Roger, "Software Engineering: A practitioner's Approach", Fifth Edition. McGraw-Hill Publisher, New York, America.)

[2] G. Rothermel and M. Harrold. "Selecting regression tests for object-oriented software," International Conference on Software Maintenance, pages 14–25, March 1994.

[3] G. Rothermel, , M.J. Harrold, "A safe, efficient regression test selection technique," ACM Transactions on Software Engineering Methodology 6(2), 173–210 (1997)

[4] H. Leung and L. White, "A firewall concept for both control-flow and data-flow in regression integration testing," In Proceedings of the Conference on Software Maintenance, pages 262–270, 1992.

[5] D. Kung, J. Gao, P. Hsia, F. Wen, Y. Toyoshima, and C. Chen. "On regression testing of object oriented programs," Journal of Systems and Software, 32(1):21–40, January 1996.

[6] Y. Jang, M. Munro, and Y. Kwon, "An improved method of selecting regression tests for C++ programs," Journal of Software Maintenance: Research and Practice, 13(5):331–350, September 2001.

[7] W. Lee, J. Khaled, and R. Brian, "Utilization of extended firewall for object-oriented regression testing," Proceedings of the 21st IEEE International Conference on Software Maintenance (ICSM'05), p 1-4.

[8] G. Rothermel, M. Harrold, and J. Dedhia. "Regression test selection for C++ software," Software Testing, Verification and Reliability, 10(2):77–109, June 2000

[9] M. J. Harrold, J. A. Jones, T. Li, and D. Liang, "Regression test selection for java software," ACM 2001 1-58113-335-9/01/10 (2010), pp 312-326.

[10] W. S. A. El-hamid, S. S. El-etriby, and M. M. Hadhoud, "Regression Test selection technique for multi-programming language," Faculty of Computer and Information, Menofia University, Shebin-Elkom, 32511, Egypt (2009).

[11] B. Árpád, et al., "Code coverage-based regression test selection and prioritization in webkit," 2012 28th IEEE International Conference on Software Maintenance (ICSM 2012), p 46-55.

[12] W. Jin, A. Orso, and T. Xie, "Automated Behavioral regression testing," 2010 Third International Conference on Software Testing, Verification and Validation, pp 137-146.

[13] N. Frechette, L. Badri, and M. Badri, "Regression Test reduction for object-oriented software: A control call graph based technique and associated tool," 2013, International Scholarly Research Network Software engineering (ISRN Software Engineering 2013), pp. 1-10.

[14] S. Horwitz, T. Reps, and D. Binkley "Interprocedural slicing using dependence graphs," ACM Transactions on Programming Languages and Systems, 12(1), January 1990, P 26–60.

[15] [15]. L. Larsen, and M. Harrold, "Slicing object-oriented software," In Proceedings of 18th IEEE International Conference on Software Engineering, (1996) 10.1109/ICSE.1996.493444, p 495-505.

[16] C. Panigrahi, and R. Mall, "An approach to prioritize regression test cases of object-oriented programs," JCSI Trans ICT (Springer) 2013, doi:10.1007/s40012-013-0011-7.

[17] C. Panigrahi, and R. Mall, "A heuristic-based regression test case prioritization approach for object-oriented programs," *Innovations in Systems and Software Engineering* (Springer) *2013*, doi: 10.1007/s11334-013-0221-z.

Samaila Musa was born in Talata Mafara, Nigeria, in 1973. He holds a MSc in Computer Science from Bayero University Kano(BUK), Nigeria in 2009. His research focus is in Software engineering particularly Software Maintenance and Search-based Software Engineering (SBSE). Samaila Musa currently is a PhD student in Information System Department, Faculty of Computer Science and Information Technology, UPM

Abu Bakar Md. Sultan was born in Melaka, Malaysia in 1965. He holds a PhD in Artificial Intelligence from University Putra Malaysia (UPM) in 2007. His research focus is in Artificial Intelligence and Software Engineering particularly Search-based Software Engineering (SBSE). He has published articles in conferences and various journals related to SBSE. Associate Professor Dr Abu Bakar Md. Sultan currently is the Dean of Faculty of Computer Science and Information Technology, UPM

Salmi Baharom. Salmi Baharom received the PhD degree in computer science from the Universiti Kebangsaan Malaysia, Malaysia in 2010. She is a senior lecturer at the Information System Department, Faculty of Computer Science and Information Technology, Universiti Putra Malaysia. Her research interests include specification-based testing, software engineering education and database design.

Abdul Azim Abd Ghani received the BSc in mathematics/computer science from Indiana State University in 1984 and MSc in computer science from University of Miami in 1985.
He received the PhD in software engineering from University of Strathclyde in 1993. He is a Professor in the Faculty of Computer Science and Information Technology, Universiti Putra Malaysia, Malaysia