TELKOMNIKA, Vol.17, No.2, April 2019, pp.844~851
ISSN: 1693-6930, accredited First Grade by Kemenristekdikti, Decree No: 21/E/KPT/2018
DOI: 10.12928/TELKOMNIKA.v17i2.10332

Regression test selection model: a comparison between ReTSE and pythia

Amir Ngah*[1], Malcolm Munro[2], Zailani Abdullah[3], Masita A.Jalil[4], Mohamad Abdallah[5]
[1,4]School of Informatics and Applied Mathematics, Universiti Malaysia Terengganu (UMT), 21030 Kuala Terengganu, Malaysia
[2]Department of Computer Science, Durham University, DL1 3LE Durham, UK
[3]Centre of Computing and Informatics, Universiti Malaysia Kelantan, Kota Bharu, Malaysia
[5]Faculty of Science and IT, Az-Zaytoonah University of Jordan, Amman, Jordan
*Corresponding author, e-mail: amirnma@umt.edu.my[1], malcolm.munro@durham.ac.uk[2], zailania@umk.edu.my[3], masita@umt.edu.my[4], m.abdallah@zuj.edu.jo[5]

Abstract

As software systems change and evolve over time regression tests have to be run to validate these changes. Regression testing is an expensive but essential activity in software maintenance. The purpose of this paper is to compare a new regression test selection model called ReTSE with Pythia. The ReTSE model uses decomposition slicing in order to identify the relevant regression tests. Decomposition slicing provides a technique that is capable of identifying the unchanged parts of a system. Pythia is a regression test selection technique based on textual differencing. Both techniques are compare using a Power program taken from Vokolos and Frankl's paper. The analysis of this comparison has shown promising results in reducing the number of tests to be run after changes are introduced.

Keywords*: decomposition slicing, program slicing, pythia, regression test selection, regression testing, ReTSE*

1. Introduction

There are many techniques have been proposed in regression testing racently [1-11]. The most important issue in regression testing is how to minimize the reuse of the existing test cases for the modified program. One of the techniques to tackle this issue is called regression test selection. This technique attempts to reduce the cost of regression testing by selecting appropriate test cases using information from the certified program, the modified program and the existing test suite [12]. Previous regression test selection approaches used an inclusive technique. Inclusive means the techniques select test cases directly from test suite that are needed in regression testing.

Regression test selection techniques can be divided into categories based on the elements used such as control-flow based [1], code entities based [13], textual differencing based [14, 15], and program slicing based [16-19]. Rothermel and Harrold [12] proposed a safe and efficient regression test selection technique based on control-flow graphs (CFG). They developed two main algorithms based on intraprocedural and interprocedural test selection algorithms. The intraprocedural algorithm operates on individual procedures and the interprocedural operates on entire programs or subsystems. In this technique, both the original and modified program will be transformed into a CFG in order to perform a comparison. The comparison algorithm will compare each node in both CFGs. If nodes differ, the algorithm will select tests from an existing test suite that exercised the node. Chen et al. [9] propose a regression test selection technique based on identifying modified code entities such as functions, variables, types, and macros. Test cases that have traversed the modified code entities will be counted in the test suite for the modified program. Vokolos and Frankl [14, 15] have developed a tool called Pythia that is used to reduce the cost of regression testing. The Unix-based tool implements an analysis technique that is called textual differencing and works by comparing the source files from the certified and modified programs. The Pythia tool can be used to analyse software systems written in the C programming language.

Received June 11, 2018; Revised November 8, 2018; Accepted December 19, 2018

There are a number of regression test selection techniques based on program slicing techniques. Binkley [16] conducted a survey about the application of program slicing to regression testing. He divided the technique into three groups of program slicing that are used in regression testing. The first group uses dynamic slicing, the second group uses program dependent graphs (PDG), and the third group is based on Weiser's data flow definition of slicing [20]. Agrawal et al. [18] have proposed three algorithms to be used in their technique called incremental regression testing. The algorithms are an execution slice, a dynamic slice, and a relevant slice. The execution slice of the program with respect to a test case is referred to as the set of statements executed under that test case. The dynamic program slice with respect to the output variables gives the statements that are not only executed but also have an effect on the program output under that test case. The relevant slice with respect to the program output for a test case is referred to the set of statements that, if modified, may alter the program output for the given test case.

Gupta et al. [19] have developed a data flow based regression testing technique that uses slicing algorithms to explicitly determine the affected definition-use associations made by a program change. The technique uses two slicing algorithms to detect directly and indirectly affected def-use associations. The first algorithm works backward from the changed statement to its definitions. The second algorithm is a forward walk from the same point as the first algorithm. The forward algorithm detects uses, and subsequent definitions and uses, which are affected by a definition that is changed at that point. Vedpal and Chaulan [21] have proposed a regression test selection technique based on identification of affected paths, affected functions and dynamic slicing which can be used to reduce the number of test cases for regression testing. This technique has two main processes. The first process is the construction of the OPDG (Object Oriented Program Dependency Graphs) for the modified program. The second process is a selection of test cases based on an algorithm that involved dynamic slicing. Qu et al. [22] have proposed a configuration selection approach for regression testing. Their technique addresses the solution to the redundancy issue in retest-all approach for configurable systems. The technique has employed an existing static slicing tool called *CodeSurfer* for computing the configurable option impact and the code change impact.

Xing et al. [23] have developed a new regression test selection technique based on the program dependence graphs of the original program and its modified version. The technique has embedded slicing technique in their two main steps. Comparing with previous techniques, they claimed their technique can eliminate some unnecessary tests to rerun. All these slicing based RTS techniques are classified as inclusion techniques which select test cases from the test suite that are needed in regression testing. The idea of the regression test selection by exclusion was proposed by Gallagher et al. [24]. Ngah et al. [25] have developed a new regression test selection by exclusion using decomposition slicing called ReTSE. Exclusion technique omits test cases from test suite that are not needed in regression testing. This paper presents the comparison between ReTSE model and Pythia technique.

2. ReTSE Model

The ReTSE model [25] is a four phase model that has three inputs and three outputs. The inputs are: the Certified Program (C); the Modified Program (M); and the existing Test Suite (TS) which includes test cases and associated test histories. The outputs are: a set of Excluded Tests (ET); a set of Optimised Regression Tests (RTO); and notification that new test cases are required. The Phases of the model are: Program Analysis, Comparison, Exclusion and Optimisation.

Phase 1 has two main steps which are Pretty Print Step and Slicing Step. This phase will produce decomposition slices for both Certified Program (C) and Modified Program (M). In Phase 2, the decomposition slices will be compared in order to produce a set of Similar Decomposition Slices (S), Different Decomposition Slices (D), Deleted Decomposition Slices (L) and New Decomposition Slices (N). In Phase 3, any test cases that not related to the D will be excluded from test suite. Finally in the Phase 4, the model will identify any redundant test cases that may exist in regression tests produced in Phase 3. More details explaination about the ReTSE model can be founded in [25].

3. Results and Analysis

Vokolos and Frankl [14] have used Power program to illustrate the Pythia, a textual differencing technique. The program aims to raise a floating point number to an integer power, using Dijkstra's algorithm. The program consists of two files: main.c and power.c. Each file contains one function. The old version of the power program (power.c) is shown in Figure 1. Because the ReTSE model only concentrates on intraprocedural, it is assumed that the changes only occur in the power.c. The new version of power function (power-v1.c) is shown in Figure 2. The old version is called Certified Program (C) and the new version is called Modified Program (M) in the ReTSE model.

```
        double power(x, n)
        double x;
        register int n;
        {
S1          int recip, sgn;
S2          double y;
S3          if (n < 0)
            {
S4              recip = 1;
S5              n = -n;
            }
            else
            {
S6              recip = 0;
            }
S7          sgn = 1;
S8          if (x < 0.0e0)
            {
S9              x = -x;
            }
S10         for (y = 1.0e0; n > 0; --n)
            {
S11             while (n % 2 == 0)
                {
S12                 x *= x;
S13                 n /= 2;
                }
S14             y *= x;
            }
S15         if (recip != 0 && y != 0.0e0)
            {
S16             return (sgn*1.0e0/y);
            }
            else
            {
S17             return (sgn*y);
            }
        }
```

Figure 1. Certified Program (C)-power.c

```
        double power(x, n)
        double x;
        register int n;
        {
S1'         int recip, sgn;
S2'         double y;
S3'         if (n < 0)
            {
S4'             recip = 1;
S5'             n = -n;
            }
            else
            {
S6'             recip = 0;
            }
S7'         sgn = 1;
S8'         if (x < 0.0e0)
            {
S9'             x = -x;
S10'            if (n % 2 == 1)
                {
S11'                sgn = -1;
                }
            }
S12'        for (y = 1.0e0; n > 0; --n)
            {
S13'            while (n % 2 == 0)
                {
S14'                x *= x;
S15'                n /= 2;
                }
S16'            y *= x;
            }
S17'        if (recip != 0 && y != 0.0e0)
            {
S18'            return (sgn*1.0e0/y);
            }
            else
            {
S19'            return (sgn*y);
            }
        }
```

Figure 2. Modified Program (M)-power-v1.c

Vokolos and Frankl used five test cases (TCi) to test the Certified Program (C). The input and output values for each test cases are shown in Table 1. Test case TC2 in Table 1 catches an error in the certified program. The THi of each TCi for C (power.c program) is shown in Table 2. In their model, THi is called a basic block execution trace which is an execution trace of a test case based on a basic block concept. A basic block is a sequence of consecutive statements with the property that control enters at the beginning statements and may leave only at the very last statement [14]. However, the THi in Table 2 is slightly different from their paper in order to consider a statement based used in the ReTSE model. The X symbol in Table 2 shows that the statement is executed for that TCi. The - symbol is for statement not executed.

These programs and information are used to analyse the ReTSE model based on their 4 phases.

Table 1. Test Suite for Certified Program (Power Program)

Test Case (TC*i*)	Input	Output
TC1	-5.0^{2}	25
TC2	$-3,0^{5}$	27
TC3	2.0^{0}	1
TC4	1.0^{4}	1
TC5	0.0^{-1}	0

Table 2. Test History (THi) of TCi for Certified Program (power.c)

Statement	TH1	TH2	TH3	TH4	TH5
S1	X	X	X	X	X
S2	X	X	X	X	X
S3	X	X	X	X	X
S4	-	-	-	-	X
S5	-	-	-	-	X
S6	X	X	X	X	-
S7	X	X	X	X	X
S8	X	X	-	-	-
S9	X	X	-	-	-
S10	X	X	X	X	X
S11	X	X	-	X	X
S12	X	X	-	X	-
S13	X	X	-	X	-
S14	X	X	-	X	X
S15	X	X	X	X	X
S16	-	-	-	-	-
S17	X	X	X	X	X

3.1. Phase 1: Program Analysis

The Original Certified Program and the Original Modified Program of the power program are assumed to have gone through the Pretty Print Step. The outputs of this step are a Certified Program (C) and Modified Program (M) as shown in Figure 2 and Figure 3 respectively. The M has two additional new statements at S10' and S11' in order to tackle the problem raised in the Certified Program (C).

In the Slicing Step, both C and M are decomposed into decomposition slices corresponding to their variables in the programs. Therefore, both programs have five decomposition slices corresponding to five variables which are *x*, *recip*, *n*, *sgn* and *y*. Decomposition slices for variables *sgn* and *y* are similar as shown in Figure 4. The output summary of the Slicing Step is given below:

DS-C = {DS-Cx, DS-Crecip, DS-Cn, DS-Csgn, DS-Cy} DS-M = {DS-Mx, DS-Mrecip, DS-Mn, DS-Msgn, DS-My}

```
[L1] double power(x, n)
     double x;
     register int n;
     {
S3     if (n < 0)
       {
S5       n = -n;
       }
S10    for(y=1.0e0; n>0; --n)
[L10]  {
S11      while (n % 2 == 0)
         {
S13        n /= 2;
         }
       }
[L16]}
```

(a) DS-C_n

```
[L1] double power(x, n)
     double x;
     register int n;
     {
S3'    if (n < 0)
       {
S5'      n = -n;
       }
S8'    if (x<0.0e0)
[L10]  {
S10'     if(n%2 == 1)
       }
S12'   for(y=1.0e0; n>0; --n)
       {
S13'     while (n % 2 == 0)
         {
S15'       n /= 2;
         }
       }
[L20]}
```

(b) DS-M_n

Figure 3. Decomposition Slice for Variable n (Power Program)

3.2. Phase 2: Comparison

In the first part of the Comparison Phase, the decomposition slices in the DS-C are compared to the decomposition slices in the DS-M using the *diff* tool. There is no output produced from the *diff* tool for the comparison between DS-Cx and DS-Mx. Similar result was achieved in the comparison between DS-Crecip and DS-Mrecip. Therefore, these decomposition slices are included in a set of pairs of Similar Decomposition Slice (S). An output is produced from the *diff* tool for the comparison between DS-Cn and DS-Mn as shown in Figure 3. Therefore, both decomposition slices are included in a set of pairs of Difference Decomposition Slice (D). The DS-Csgn and DS-Msgn as shown in Figure 4 are also included in the D because the output is produced from the *diff* tool for this comparison. The DS-Cy and DS-My in Figure 4 are also included in the D because the *diff* tool also produces an output from this comparison. The output summary of the first part of the Comparison Phase is shown in Table 3.

```
   double power(x, n)
   double x;
   register int n;
   {
S1   int recip, sgn;
S2   double y;
S3   if (n < 0)
     {
S4     recip = 1;
S5     n = -n;
     }
     else
     {
S6     recip = 0;
     }
S7   sgn = 1;
S8   if (x < 0.0e0)
     {
S9     x = -x;
     }
S10  for(y=1.0e0; n>0; --n)
     {
S11    while (n % 2 == 0)
       {
S12      x *= x;
S13      n /= 2;
       }
S14    y *= x;
     }
S15  if(recip!=0 && y!=0.0e0)
     {
S16    return (sgn*1.0e0/y);
     }
     else
     {
S17    return (sgn*y);
     }
   }
```

(a) DS-C$_{sgn}$/DS-C$_y$

```
   double power(x, n)
   double x;
   register int n;
   {
S1'  int recip, sgn;
S2'  double y;
S3'  if (n < 0)
     {
S4'    recip = 1;
S5'    n = -n;
     }
     else
     {
S6'    recip = 0;
     }
S7'  sgn = 1;
S8'  if (x < 0.0e0)
     {
S9'    x = -x;
S10'   if (n % 2 == 1)
       {
S11'     sgn = -1;
       }
     }
S12' for(y=1.0e0; n>0; --n)
     {
S13'   while (n % 2 == 0)
       {
S14'     x *= x;
S15'     n /= 2;
       }
S16'   y *= x;
     }
S17' if(recip!=0 && y!=0.0e0)
     {
S18'   return (sgn*1.0e0/y);
     }
     else
     {
S19'   return (sgn*y);
     }
   }
```

(b) DS-M$_{sgn}$/DS-M$_y$

Figure 4. Decomposition Slice for Variable sgn/y (Power Program)

Table 3. Comparison Results between DS-Cvi and DS-Mvi (Power Program)

Set of	Member of Set
D	{(DS-C*n*, DS-M*n*), (DS-C*sgn*, DS-M*sgn*), (DS-C*y*, DS-M*y*)}
S	{(DS-C*x*, DS-M*x*), (DS-C*recip*, DS-M*recip*)}
S	{}
L	{}

The second part of the Comparison Phase is a more detailed comparison between DS-Cv_i and DS-Mv_i only if they are members of D. In this case, only the decomposition slices for variables *sgn* (DS-C$_{sgn}$ and DS-M$_{sgn}$), *n* (DS-C$_n$ and DSM$_n$) and *y* (DS-C$_y$ and DS-M$_y$) areinvolved in the second part of the comparison because they are members of D (shown in the first part of the comparison). The comparison output using the *diff* tool is given below:

a. Comparison between DS-Csgn and DS-Msgn 19a20,23
> if (n % 2 ==1)
> {
> sgn = -1;
> }
b. Comparison between DS-Cn and DS-Mn 8a9,12
> if (x < 0.0e0
> {
> if (n % 2 ==1)
> }
c. Comparison between DS-Cy and DS-My 19a20,23
> if (n % 2 ==1)
> {
> sgn = -1;
> }

In the comparison between DS-C$_{sgn}$ and DS-M$_{sgn}$ (Figure 4), the statement at line 19 ([L9]) from DS-C$_{sgn}$ is included in the set of Change Statement for variable *sgn* (CS$_{sgn}$). Any statement in the range of lines 20 ([L20]) to 23 ([L23]) from DS-M$_{sgn}$ is included in the set of Relevant Change statement for variable *sgn* (RCS$_{sgn}$). Therefore, S9 from DS-C$_{sgn}$ is included in the CS$_{sgn}$ and statements S10' and S11' from DS-M$_{sgn}$ are included in the RCS$_{sgn}$. Statement S9' is also included in the the RCS$_{sgn}$ because it is located at the same branch of statement S10'. The same happens to the comparison between DS-C$_y$ and DS-M$_y$ as shown in Figure 4 where statement S9 from DS-C$_y$ is included in the CS *y* and statements S9', S10' and S11' from DS-M$_y$ are included in the RCS$_y$.

In the comparison between DS-C$_n$ and DS-M$_n$ as shown in Figure 3, line 8 ([L8]) is not a statement but a close curly bracket (*}*). Therefore, the statement immediately after that symbol will be included in the set of Change Statement for variable *n* (CS$_n$). Any statement in the range of lines 9 ([L9]) to 12 ([L12]) from DSM$_n$ is included in the set of Relevant Change Statement for variable *n* (RCS$_n$). Therefore, statement S10 is included in the CS$_n$ and statements S8' and S10' are included in the RCS$_n$. Then the CS is produced from the union of CS$_{sgn}$, CS$_y$ and CS$_n$ where the RCS is produced from the union of RCS$_{sgn}$, RCS$_y$ and RCS$_n$. A summary of the second part of the Comparison Phase is given below:

CS = CSsgn U CSy U CSn
= {S9} U {S9} U {S10}= {S9, S10} RCS= RCSsgn U RCSy U RCSn
= {S9', S10', S11'} U {S9', S10', S11'} U {S8', S10'}
= {S8', S9', S10', S11'}

3.4. Phase 3: Exclusion

There are five test cases (TC$_i$) that were used by Vokolos and Frankl for Certified Program (C) of power.c program as shown in Table 1. Their TH$_i$ is shown in Table 2. Any TC$_i$ where the CS is not subset of TH$_i$, will be included in the set of Excluded Test (ET). In this case, the CS which includes statements S9 and S10, is not subset of TH3, TH4 and TH5. Therefore, TC3, TC4, and TC5 will be included in the set of ET. The remaining test cases in Test Suite are included in the set of Regression Tests (RT). Therefore, TC1 and TC2 are included in the RT. A summary of the Exclusion Phase is given below:

RT= {TC1, TC2}

3.5. Phase 4: Optimisation

In the Optimisation Phase, all TC*i* that are members of the RT will be executed onto UDS-M. In this case, there are only three DS-M*vi* members of D which are DS-M*sgn*, DS-M*n* and DS-M*y*. The union of these decomposition slices (UDS-M) has produced the same program as M (power-v1.c) as shown in Figure 2 which includes statements S1', S2', S3', S4', S5', S6', S7', S8', S9', S10', S11', S12', S13', 14', S15', S16', S17', S18' and S19'. The RCS produced in the second part of the Comparison Phase is used here. The RCS includes statements S8', S9', S10' and S11'.

Test cases in RT, which are TC1 and TC2, are sequently executed onto the UDS-M. Firstly, TC1 is executed onto UDS-M. The RTE1 for TC1 is S1', S2', S3', S6', S7', S8', S9', S10', S11', S12', S13', S14', S15', S16', S17' and S19'. The RTE1 contains all members of RCS. That means the RCS receive full coverage by executed only the TC1. The execution of test cases is stopped because the RCS has already achieved full coverage. This means it is enough to use only TC1 as a regression test for the modified program. Therefore, TC1 will be included in the RTO. The remaining test case TC2 will be ignored for RTO. A summary of the Optimisation Phase is given below:

UDS-M = {S1', S2', S3', S4', S5', S6', S7', S8', S9', S10', S11', S12', S13', S14', S15', S16', S17', S18', S19'}
RCS = {S8', S9', S10', S11'}
TC1,
RTE1= {S1', S2', S3', S6', S7', S8', S9', S10', S11', S12', S13', S14', S15', S16', S17', S19'}
RCS-current= {S8', S9', S10', S11'}
RCS-coverage = {S8', S9', S10', S11'} RTO= {TC1}
RCS-full= YES
Therefore, the final output of the model for this case is given below: RTO = {TC1}
Request_New_Test_Cases = NO

4. Conclusion

The Pythia selected two test cases from an existing test suite to test a new version of the power program [10]. The same program has been applied to the ReTSE model. In the Exclusion Phase, the ReTSE model has selected the same two test cases as the Pythia technique. Moreover, after Optimisation Phase, the ReTSE model has selected only one test case to test a new version of the power program. This is because the model has identified that both test cases (produced in the Exclusion Phase) are redundant at the same coverage of a new version of the program.

References

[1] Binh NT, Duy TC, Parissis. LusRegTes: A Regression Testing Tool for Lustre Programs. *International Journal of Electrical and Computer Engineering (IJECE)*. 2017; 6(5): 2635-2644.
[2] Ngah A, Munro M, Abdallah M. An Overview of Regression Testing. *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)*. 2017; 9(3-5): 45-49.
[3] Pandey A, Banerjee S. Test suite minimization in regression testing using hybrid approach of ACO and GA. *International Journal of Applied Metaheuristic Computing.* 2018; 9(3): 88-104.
[4] Chen L, Zhang L. *Speeding up mutation testing via regression test selection: An extensive study*. Proceedings of the 11th IEEE International Conference on Software Testing, Verification and Validation (ICST 2018). 2018: 58-69.
[5] Jimenez I, Watkins N, Sevilla M, Lofstead J, Maltzahn C. *Quiho: Automated performance regression testing using inferred resource utilization profiles*. Proceedings of the ACM/SPEC International Conference on Performance Engineering. 2018: 273-284.
[6] Minhas NM, Petersen K, Ali N, Wnuk K. *Regression testing goals-view of practitioners and researchers*. Proceedings of the 2[4th] Asia-Pacific Software Engineering Conference Workshops (APSECW 2017). 2018: 25-31.
[7] Tulasiraman M, Kalimuthu V. Cost cognizant history based prioritization of test case for regression testing using immune algorithm. *International Journal of Intelligent Engineering and Systems.* 2018; 11(1): 221-228.

[8] Lingming Zhang. *Hybrid regression test selection*. Proceedings of the IEEE/ACM 40th International Conference on Software Engineering (ICSE 2018). 2018: 199-209.
[9] Annibale Panichella, Rocco Oliveto, Massimiliano Di Penta, Andrea De Lucia. Improving Multi- Objective Test Case Selection by Injecting Diversity in Genetic Algorithms. *IEEE Transactions on Software Engineering*. 2015; 41(4): 358-383.
[10] Sumit Dahiya, Rajesh K.Bhatia, Dhavleesh Rattan. Regression Test Selection using Class, Sequence, and Activity Diagrams. IET Software. 2016; 10(3): 72-80.
[11] Seyedeh Sepideh Emam, James Miller. Test Case Prioritization using Extended Digraphs. ACM Transactions on Software Engineering and Methodology (TOSEM). 2015; 25(1): 6:1-6:41.
[12] Rothermel G, Harrold MJ. A Safe, Efficient Regression Test Selection Technique. *ACM Transactions on Software Engineering Methodology (TOSEM)*. 1997; 6(2): 173–210.
[13] Chen YF, Rosenblum DS, Vo KP. *Testtube: A System for Selective Regression Testing*. Proceedings of the International Conference on Software Engineering (ICSE'94). 1994: 211–220.
[14] Vokolos FI, Frankl PG. *Pythia: A Regression Test Selection Tool Based on Textual Differencing*. Proceedings of the International Conference on Reliability, Quality and Safety of Software-intensive Systems (ENCRESS'97). 1997: 3–21.
[15] Vokolos FI, Frankl PG. *Empirical Evaluation of the Textual Differencing Regression Testing Technique*. Proceedings of the International Conference on Software Maintenance (ICSM'98). 1998: 44–53.
[16] Binkley D. The Application of Program Slicing to Regression Testing. *Information & Software Technology*, 1998; 40(11-12): 583–594.
[17] Forgács I, Hajnal A, Takács É. *Regression Slicing and Its Use in Regression Testing*. Proceedings of the IEEE International Computer Software and Applications Conference (COMPSAC'98). 1998: 464–469.
[18] Agrawal H, Horgan JR, Krauser EW, London SA. *Incremental Regression Testing*. Proceedings of the International Conference on Software Maintenance (ICSM'93). 1993: 348–357.
[19] Gupta R, Harrold MJ, Soffa ML. *An Approach to Regression Testing Using Slicing*. Proceedings of the International Conference on Software Maintenance (ICSM'92). 1992: 299–308.
[20] Weiser M. *Program Slicing*. Proceedings of the International Conference on Software Engineering (ICSE'81). 1981: 439–449.
[21] Vedpal, Chauhan N. *Regression Test Selection for Object Oriented Systems using OPDG and Slicing Technique*. Proceedings of the 2nd International Conference on Computing for Sustainable Global Development (INDIACom). 2015: 1372-1378.
[22] Qu X, Acharya M, Robinson B. *Configuration Selection Using Code Change Impact Analysis for Regression Testing*. Proceedings of the 28th IEEE International Conference on Software Maintenance (ICSM). 2012; 129-138.
[23] Xing J, Wang H, Song W, Yang Q. *Safe Regression Test Selection Based on Program Dependence Graphs*. Proceedings of the 36th IEEE Annual Computer Software and Applications Conference Workshops (COMPSACW). 2012; 188-193.
[24] Gallagher K, Hall T, Black S. *Reducing Regression Test Size by Exclusion*. Proceedings of the International Conference on Software Maintenance (ICSM'07). 2007; 154–163.
[25] Ngah A, Saman MY, Munro M. ReTSE: Slicing based Regression Testing. *WIT Transactions on Engineering Sciences*. 2014; 86: 457-469.