False Test Case Selection : Improvement of Regression Testing Approach

Benjawan Srisura
Information Technology Department
Vincent Mary School of Science and Technology
Assumption University, Thailand
benjawan@scitech.au.edu

Adtha Lawanna
Information Technology Department
Vincent Mary School of Science and Technology
Assumption University, Thailand
adtha@scitech.au.edu

***Abstract*—Regression testing has been considered as a time-consumed process in software testing. In a recent year, one of interesting research work initiated for minimizing testing time is finding a technique in selecting test cases from a large test suit. Most of test cases selection technique in literature considers test cases that are related to the requirement's changed. During executing test cases that are related to the modified part, a set of fail test case is accidentally emerged and make test suit has become larger. Therefore, this paper proposes a technique in selecting suitable false test cases when they are generated in regression testing. However, in order to ensure that the quality and validity of using the proposed technique are acceptable, an experiment was systematically conducted in this study. And we also found that the false test case selection technique can minimize the size of test suit, effectively. (*Abstract*)**

Keywords—Test case selection; Regression Test Case Selection; Regression Testing; Sotware Testing (key words)

I. INTRODUCTION

One of the important phases in the software development systems is software testing [1]. It evaluates the software's capability and reveals errors before delivering the desired software to users. Generally, regression testing is a repeating process which is executed in software testing to confirm the requirements have been implement correctly and also ensure that any applied bug's fix or a change of code have not affected previous functionalities and introduced any errors.

Generally, regression testing approach can be separated into three main processes - test case minimization, test case selection and test case prioritization. Test case prioritization process considers ordering of test cases for detection of faults at the earliest while test case minimization is carried out to eliminate the redundant test cases. Finally, test case selection process, its aim is to be able to select the subset of test cases from the test suite that has the potential to detect errors initiated from a change.

For a large scale software product, there are many numbers of test cases available in its test suite. Whenever a new version of software is released, re-running all test cases is executed with a potentially cost, time and effort [2, 3]. Therefore, a need of selecting a suitable subset of test cases from a large test suite has become an interesting issue to be improved.

Interestingly, considering the control flow of program module in selecting appropriated test cases has been concerned as high safe and precise techniques in the literature [4, 5, 6, 7, 8]. Unfortunately, in every test case execution, fail test cases set have been generated and then they will be totally delivered into the test suit for retesting next cycle, automatically. This situation may produce a test suit whose size is constant or become bigger.

As we consider the various types of delivered false test cases, each of them has its own level of defecting – such as some are effects from another, some are critical error. However, this information has never been recognized for improving consecutive test case selection. If considering false test case can be used to be another criterion in selecting the suitable test cases, test suit size may be minimized, significantly. Therefore, this paper proposes a false test case selection technique for supporting regression test selection in considering suitable test cases for the next execution.

II. RELATED WORKS

According to the previous studies, this paper identifies pattern of regression test selection (RTS) techniques into two main approaches. The first approach, selecting test case with non-criteria – i.e. experts' judgement which tends to become unreliable with various type of software product, retest-all (AS) technique whose performance is quit poor due to the high cost of executing entire test cases, random selection technique that is ineffective in a large software scale.

Secondly, at least one criterion is carried out to be considered for selecting suitable test cases. RTS techniques of this group can be categorized based on how to identify test cases that affect to the modifications' part – including Dataflow technique which analyzes the data flow or data path of program's module [9, 10, 11] which is unsafe and imprecise because it does not consider the control dependency among program element, Slicing techniques [12, 13] selects test cases that generate different outputs which is precise because it does not involve test cases which do not produce different outputs, Firewall techniques [14, 15] select only test cases execute the modified module which may unsafe and imprecise, Differencing techniques (such as detecting code or textual difference) [16, 17] whose precision may poor, and Control flow technique [8] analyze control flow of modified module to

978-1-4673-9749-0/16/$31.00 ©2016 IEEE

find all relevant test cases which has been found that is most acceptable safe and precise.

Therefore, the control flow technique has been used to be ground technique in extending the proposed idea in term of the false test case selection for reducing test suit size to be more effectively.

III. FALS TEST CASE SELECTION TECHNIQUE

The regression testing normally requires executing a test suit (T') several rounds until all test cases in T' are completely solved. The results of testing test cases are recorded in a log file. After that, "false" test cases will be entirely selected to execute in next cycle as shown in *"Fig. 1"*.

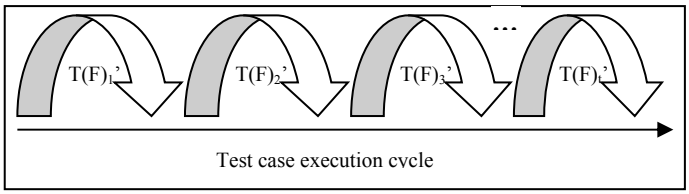

Fig. 1. Traditional regression testing execution cycle

Since the false test case has been concerned as a pivot part that all of them must be re-executed, it is not necessary to re-execute all of them in every execution time. While the safe and precision has been preserved, this paper proposes a technique to consider whether that false test case should be re-tested next cycle or not. Minimizing false test case size will produce the optimized execution time as shown in *"Fig. 2"*.

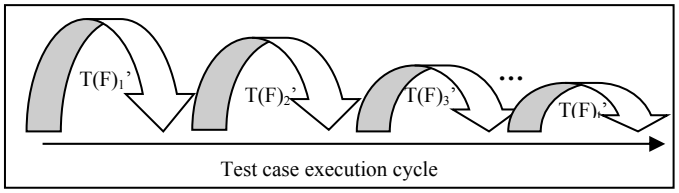

Fig. 2. Minimizing number of false test case in each execution cycle

When we consider the log file, the false test case is reported in term of the severity code standard. It can be summarized to be four main levels in the table shown below.

TABLE I. SEVERITY CODES AND DESCRIPTION

Code Name	Descriptions	Example
Low (S1)	Test does not affect functionality, data, productivity and efficiency. It is an inconvenience to use.	Using unfamiliar, grammatical error word.
Minor (S2)	Test case produces correct output when inputting invalid data.	Inputting special character is allowed which is not required.
Major (S3)	Test case produces incorrect output when inputting valid data.	The output returned from function is incorrected
Critical (S4)	Test affects functionality, data, productivity and efficiency.	Test cannot proceed further.

Guideline of selecting a suitable false test case – proposed by this paper, is to consider the level of the severity code reported in the log file.

If a false test case is reported as low (S1) level which defect notifies only warning and not affect with any program modules, it can be canceled to be selected for next execution.

In the part of minor (S2) level, a false test case will test the invalid input; however, the corrected output is returned. This can be interpreted that those program modules have provides some methods to handle invalid input. In order to ensure that relevant test case should not be ignored to select – preserving safe, a mechanism in verifying whether a method or catch will be invoked to handle this error or not is proposed.

In the contrast way, a test case returns incorrect output although a valid input is identified will be justified to be a member of major (S3). Before selecting it, a method to check the redundant test case is also provided for optimizing the false test suit size.

Finally, the critical (S4) false test case mostly affects program's productivity. Therefore, all test cases that force the program shut down must be unconditional selected and retested until they have been solved completely.

A. Conceptual Model of False Test Case Selection Technique

Regarding to the given guideline in the previous section, the conceptual model is generated for expressing all used criterions in selecting a suitable false test case clearly.

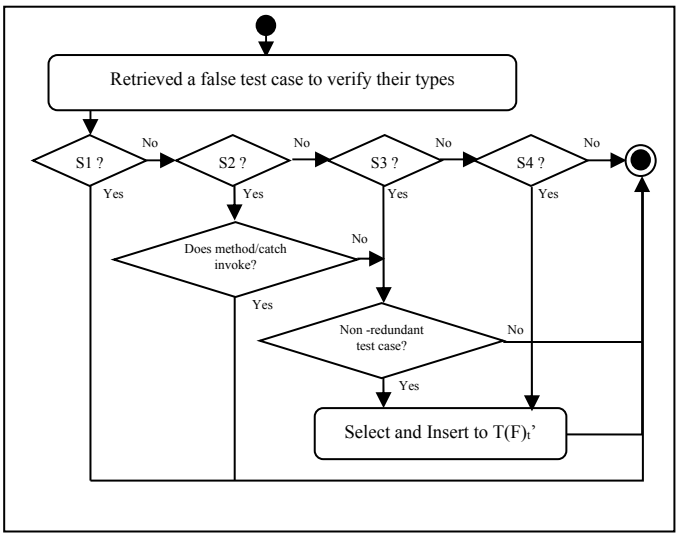

Fig. 3. Conceptual Model of False Test Case Selection

In case that severity code of each false test case is unknown, valid input and output can be used to classify the severity level, instead. Therefore, the valid input and output are optional requested to be input of the proposed technique. Moreover, the original program (P) and modified program (P') will be delivered for finding all relevant false test cases. Finally, the returned outcome will be a minimized false test suit (T(F)'). To demonstrate how the proposed technique – named as False Test Case Selection (FTS), works, the algorithm is generated and shown in *"Fig 4"*.

```
Algorithm:  SelectTestCase (P, T(F), ValidInput, ValidOutput) : T(F)'
Input:      P, P' - The original and modified program,
            T(F) - A false test case set used to test P or A false log file
Output:     T(F)' - Subset of T(F) selected for executing P'
begin
    T(F)' = ϕ
    i=1
    G = Construct ICFG from P
    ti = Read a false test from T(F)
    while NOT end of log file(T(F))
    begin
      select ti
      case S1 (ValidInput = TRUE) and (ValidOutput = TRUE):  break;
      case S2 (ValidInput = FALSE) and (ValidOutput = TRUE):
            begin
                Gti = Construct ICFG from ti
                If InvokeMethodCall(G, Gti) a
                    break;
                else  if NonRedundantTestCase (ti, T(F)' b)
                        break;
                    else  T(F)' = T(F)' + ti
            end
      case S3 (ValidInput = FALSE) and (ValidOutput = TRUE):
            begin
                if NonRedundantTestCase (ti, T(F)' b)
                    break;
                else  T(F)' = T(F)' + ti
            end
      case S4 (ValidInput = FALSE) and (ValidOutput = TRUE):
            begin
                T(F)' = T(F)' + ti
            End
      i = i+1
    end

    Return T(F)'
end
```

a. "InvokeMethodCall" method is explained in section B.

b. "NonRedundantTestCase" method is denoted in section C.

Fig. 4. False Test Case Selection Algorithm

B. Detecting Method Call

Since FTS technique has been implemented under the basis of the control flow graph concept, Inter-procedural Control Flow Graph (ICFG) [18] is adopted to verify whether a false test case (S2) invoke any method or catch or not. Therefore, "InvokeMethodCall" method is initiated to analyze ICFG of a concerned false test case whose type is S2 in two issues – Invoking overloaded method or No invoking overloaded method. To ensure that ignoring that test case invokes an overloaded method for protecting some kinds of errors or supporting various type of users' input, inputting invalid inputs to the program that can return a valid output may possibly come up and also reasonably concerned as ignorance next execution. Otherwise, S2 test case should be re-executed next cycle if it is not redundant false test case.

```
Algorithm:  InvokeMethodCall (G, Gti) :  TRUE or FALSE
Input:      G – ICFG from P contains collection of graph path
            Gti – ICFG from ti is a graph path.
Output:     TRUE when ti invoke a method, otherwise it returns FALSE
begin
    j=1
    while NOT end of (G)
    begin
      if Gti is member of G
          Return TRUE
        Else Return FALSE
    j = j+1
    end
end
```

Fig. 5. InvokeMethodCall Algorithm

C. Test Case Redundancy Dectection

Since the test suit (T(F)') that will be generated from the proposed technique should be optimized, checking redundant test case is designed to be another pivot part of this technique as well. ICFG of a false test case will be compared with every existing ICFG false test cases in T(F)'. A method – named "NonRedundantTestCase" method is designed as follows:

```
Algorithm:  NonRedundantTestCase (ti, T(F)') :  TRUE or FALSE
Input:      ti - A target false test case,
            T(F)' - Subset of T(F) selected for executing P'
Output:     TRUE when ti is not redundant path, otherwise it returns FALSE
begin
    j = 1
    Gti = Construct ICFG from ti
    tj = Read a false test j from T(F)'
    Gj = Construct ICFG from tj
    while (NOT end of (T(F)')) and (Gti <> Gj)
    begin
    j = j+1
    tj = Read a false test j from T(F)'
    Gj = Construct ICFG from tj
    End
    if (Gti == Gj)
        Return TRUE
    Else Return FALSE
end
```

Fig. 6. NonRedundantTestCase Algorithm

IV. Experimental Design and Methods

In order to provide a validity of the proposed approach and show its benefits in term of how a false test suit T(F)' can be reduced, a case study is presented in this section.

A. The Case Study

The case study is a program to find a maximum value and shown in *"Fig.7"*.

```
1 public class MaxApp {
2   public static void main(String[ ] args) {
3     int i = 1;
4     int  j = 5;
5     double k = 5.95;
6     char c = 'a';
7     System.out.println("T1: The maximum value are : " + max(i,j));
8     // System.out.println("T2 : The maximum value is : " + max(c,i));
9     // System.out.println("T3 : The maximum value is : " + max(j,k));
10    // System.out.println("T4: The maximum value is : " + max(i,j,k));
11  }
12  public static double max (int iValue, int jValue)
13  {   if (iValue > jValue)
14          return iValue;
15      else return jValue;
16  }
17  public static double max (double iValue, double jValue)
18  {  if (iValue > jValue)
19          return iValue;
20     else return jValue;
21  }
22  public static double max (char iValue, int jValue)
23  {  if (iValue > jValue)
24          return iValue;
25     else return iValue;
26  }
35 }
```

Fig. 7. Program Case Study

In order to observe and trace which false test cases will be selected to execute with the modified program (P') based on the proposed technique, the inter procedural control flow graph (ICFG) of P is generated by considering its function calls and returns.

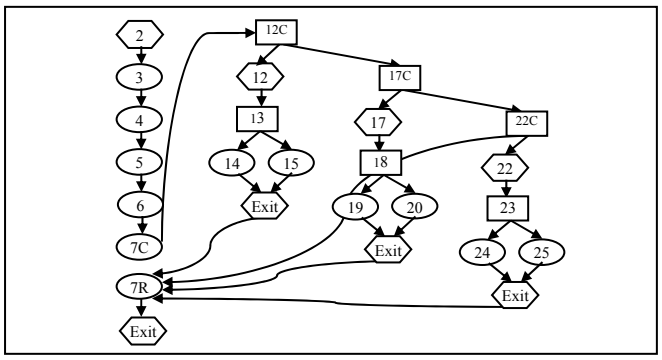

Fig. 8. Inter Procedural Control Flow of Case Study

Then, gathering edge trace for each false test case (T(F)') recorded in the log file is considered and summarized in the table shown below.

TABLE II. FALSE TEST CASE SUMMARIZATION

False Test Case	False Type	Edge Trace G_{ti}	Input	Output	Selected Test Case
t_1	S1	(2,3),(3,4),(4,5),(5,6),(6,7C),(7,12C),(12,13),(13,15),(15,7R)	√	√	X
t_{21}	S2	(2,3),(3,4),(4,5),(5,6),(6,7C),(7,17C),(17,18),(18,19),(19,7R)	X	√	X
t_{22}	S2	(2,3),(3,4),(4,5),(5,6),(6,7C),(7,7R)	X	√	√
t_{31}	S3	(2,3),(3,4),(4,5),(5,6),(6,7C),(7,7R)	√	X	X
t_{32}	S3	(2,3),(3,4),(4,5),(5,6),(6,7C),(7,22C),(22,23),(23,25),(25,7R)	√	X	√
t_4	S4	(2,3),(3,4),(4,5),(5,6),(6,...)	√	Fail	√

Suppose that there are six false test cases reported in the log file. Each of them will be transformed into edge trace (G_{ti}) for selecting based on the proposed technique. The test case t_1 - whose type is S2, is not selected because its result is always valid with grammatical or word error detection. However, two test case under type S2 are judged in different ways: test case t_{21} is not chosen due to its method invoke but test ca se t_{22} is selected because it may contain the invalid process although valid output returns. In case of type S3, t_{31} and t_{32} test case are analyzed their redundancy. If t_{32} is duplicated with existing test case, it should be removed from T(F)'. Finally, the false test case type S4 (test case t_4) which usually produces a critical problem that may force the program shut down is automatically selected to be test next execution. Therefore, there are only three from six false test cases – including t_{22}, t_{32} and t_4, are selected to be executed with P', instead.

B. Experimental Design

The comparative study had been set up to evaluate the validity and performance in term of the reduction cost of the proposed technique. The experiment collected 4 programs which were constructed from an object-oriented programming course whose minimum line of code was approximately 5,000 lines to serve the point of sale unit of a small enterprise business – including food court, parking, book store and restaurant. Each program was requested to prepare its own false test case suit (F(T)) and log file.

In order to guarantee the effectiveness of the proposed selection technique – FTS, which extends the idea of reducing the number of false test cases into the Control Flow (CFS) technique, the false test case suit (T(F)) of FTS will be compared with the T(F) of CFS technique.

Moreover, the Retest-All (AS) test case selection which is a representative from non-criteria selection group is be implemented to compare with the propose technique as well.

C. Evaluation Criteria

Refer to *"Fig.2"*, this paper tries to invent a test case selection technique which can reduce size of the false test cases (T(F)') size so criteria used to evaluated the effectiveness of the proposed technique is the percentage of test suit reduction rate [19] of the false test case which is represented as R(T):

$$R(T)\ \% = (T(F) - T(F)') / T(F) * 100 \quad (1)$$

In order to ensure that the optimization is reasonable safe - which is to make sure that all relevant test cases should be selected, and precise – which is to guarantee that all selected test cases should be relevant test cases [19], they have been used to evaluate the proposed technique as well.

V. RESULTS AND DISCUSSIONS

Due to the goal to be achieved by this study is to find a test case selection technique that can reduce size of the false test case suit (T(F)) for finally leading to a fewer execution times, the experimental results returned from tested programs are summarized as follows:

TABLE III. RESULTS OF FALSE REDUCTION RATE (AS : CFS)

Application	T(F) AS	T(F)' CFS	False Reduction Rate (%)
Food Court Cashier (FCC)	25	24	4
Parking Cashier (PC)	20	18	10
Book Store Cashier (BSC)	50	46	8
Restaurant Cashier (RC)	30	28	6

The false reduction rate between the comparative techniques - the Retest All (AS) and the Control Flow (CFS), and the proposed (FTS) techniques are summarized in tables shown above. We found that the false reduction rate of CFS technique is better than AS technique whose average false reduction rate computed from 4 experiments is around 7%. Since AS technique selects all false test cases to be re-executed next cycle, the modified test suit (T(F)') of AS is always constant. In the part of CFS technique, it chooses only the false test cases that affect to the modified part of program (P'). Therefore, the false test case suit (T(F)') size produced by CFS technique can be minimized.

TABLE IV. RESULTS OF FALSE REDUCTION RATE (AS : FTS)

Application	T(F) AS	T(F)' FTS	False Reduction Rate (%)
Food Court Cashier (FCC)	25	20	20
Parking Cashier (PC)	20	16	20
Book Store Cashier (BSC)	50	40	20
Restaurant Cashier (RC)	30	25	16

Considering the false reduction rate between AS technique and the proposed technique (FTS), it is 19% approximately. Since FTS technique tries to select the relevant (which comes from CFS technique) and unique false (which are initiated from the proposed idea) test cases, the false reduction rate of FTS is found to be better than AS technique.

TABLE V. RESULTS OF FALSE REDUCTION RATE (CFS : FTS)

Application	T(F) CFS	T(F)' FTS	False Reduction Rate (%)
Food Court Cashier (FCC)	24	20	16
Parking Cashier (PC)	18	16	11
Book Store Cashier (BSC)	46	40	13
Restaurant Cashier (RC)	28	25	10

However, we consider the average reduction rate between CFS and FTS techniques; it can be estimated 12.5% in average. It conforms to the expectation that CFS is the ground technique of FTS therefore FTS reduction rate can be computed to show its better performance. Moreover, if we compare the average reduction rates between AS:CFS (7%) with AS:FTS (19%); it can clearly summarized that performance of reducing the false test suit size (T(F)') of FTS technique is sound to be better than CFS technique.

The graph shown below shows that the false test case suit size (T(F)') which is generated from FTS technique can be optimized and it is smaller than the false test case suits (T(F)') which is produced AS and CFS technique.

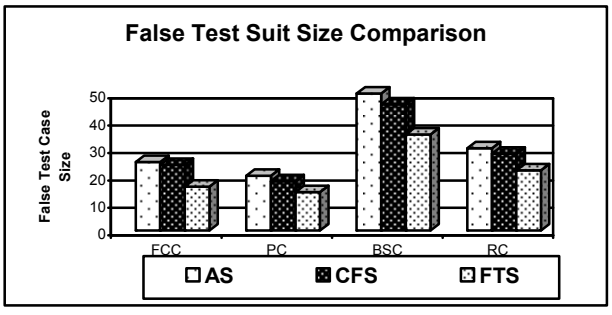

Fig. 9. Comparative False Test Case Size

Finally, evaluating the validity of the proposed technique (FTS) under the precise measure when it is used to guarantee that all selected test cases should be relevant test cases; since all false test cases is a subset relevant test case, selecting a false test case set T(F)' to be executed next cycle is meant that selecting a subset of relevant false test case as well. On the other hand, there is no non-relevance false test case will be selected into the false test suit T(F)', absolutely. This can be concluded that the precision of the proposed technique is still valid although its size is minimized as graph shown below.

Furthermore, another interesting measure used to just the validity of FTS technique is safe which is used to make sure that all relevant test cases should be selected. If we identify a false test is relevant test case no matter whose type (S1, S2, S3 and S4) is, safe of FTS technique may be a bit dropped when we compare with CFS technique. Because FTS technique selects all types of unique false test case, except S1 false test case. However, safe of FTS technique is found to be better than AS technique because AS select all test cases that may or may not relevant to the test suit T' and also the false test suit (T(F)').

Therefore, we can conclude that the false test case selection (FTS) technique contributes a benefit in reducing size of the test suit when compared with the traditional techniques. However, we have not yet guarantee that this technique can execute with the best performance, there is another type of test case - "pass" test case, has been considered as another part of the proposed technique.

For the recommended step for the advance research, both "pass" and "false" test case should be considered together for developing a better safe selection technique in the regression testing. Moreover, the scale of programs or software products used to be tested should become larger than this study in order to ensure that the effectiveness of the proposed technique in the complicated environment has been preserved.

REFERENCES

[1] R. Pressman, "Software Engineering: A Practitioner's Approach", McGraw-Hill, New York, 2002.

[2] H. Leung and L. White, "Insights into regression testing", Proceedings of the Conference on Software Maintenance, pages 60–69, 1989.

[3] G. Kapfhammer, "The Computer Science Handbook, chapter on Software testing", CRC Press, Boca Raton, FL, 2nd edition, 2004.

[4] G. Rothermel and M. Harrold. "Selecting tests and identifying test coverage requirements for modified software", Proceddding of the International Symposium on Software Testing and Analysis, p. 169-184, august 1994.

[5] Rothermel G, Harrold MJ, "A safe, efficient algorithm for regression test selection", Proceedings of International Conference on Software Maintenance (ICSM 2003), IEEE Computer Society Press, 1993, p. 358–367.

[6] J. Laski and W. Szermer "Identification of program modifications and its applications in software maintenance", In Proceedings of the Conference on Software Maintenance, November 1992.

[7] A. Ali, A. Nadeem, Z. Iqbal, and M. Usman. "Regression testing based on UML design models", In Proceedings of the 13th Pacific Rim International Symposium on Dependable Computing, 2007.

[8] G. Rothermel and M. Harrold. "A safe, efficient regression test selection technique", ACM Transactions on Software Engineering and Methodology, 6(2): 173-210, April 1997.

[9] Harrold MJ, Soffa ML “Interprocedual data flow testing” , Proceedings of the 3rd ACM SIGSOFT Symposium on Software Testing, Analysis, and Verification (TAV3), ACM Press, 1989.

[10] M. Harrold and M. Soffa, “An incremental approach to unit testing during maintenace”, Proceedings of the International conference on Software Maintenace, p. 362-367, October 1988.

[11] M. Harrold and M. soffa, “Interprocedural data flow testing”, Proceedings of the ACM SIGSOFT ’89 third symposium on Software testing, analysis and verification, p. 158-167, December 1989.

[12] H. Agrwal, J. Horgan, E. Krauser, and S. London, “Incremental regression testing”, IEEE International Conference on software Maintenance, p.348-357, 1993.

[13] D. Binkley, “Semantics guid regression test cost reduction”, IEEE Transactions on Software Engineering, 23(8):498-516, August 1997.

[14] H. Leung and L. White, “A study of integration testing and software regression at the integration level”, Procedding of Conference on Software Maintenance, p290-300, November 1990.

[15] H. Leung and L. White, “A firewall concept for both control-flow and data-flow in regression integration testing”, Proceeding Conference on Software Maintenace, p.262-270, 1992.

[16] Y. Chen, D. Rosenblum, and K. Co., “TestTube: A system for selecting regression testing”, Procedding of the 16th International Conference on Software Engineering, p.211-222, May 1994.

[17] F. Vokolos and P. Frankl. “Pythia: A Regression tes selection tool based on textual differencing”, Proceddings of the 3rd International Conference on Reliability, Quality abd Safety of Software-Intensive System (ENCRESS’ 97), p.3-21, May 1977.

[18] F. Nielson, H. R. Nielson, “Interprocedural Control Flow Analysis”, 8th European Symposium on Programming (ESOP’99) the Joint European Conferences on Theory and Proactice of Software, p.20-39, March 1999.

[19] G. Rothermel and M. Harrold, “Analyzing regression test selection techniques”, IEEE Transactions on Software Engineering, p. 22(8):529-551, August 1996