The Improvement of Test Case Selection for the Process Software Maintenance

Adtha Lawanna*

Abstract

Software maintenance is one of the critical parts in the processes of Software-Development Life Cycle. The body knowledge of maintaining the new software versions base on the abilities of fixing bugs, the revision of programs, the plan of preventing future constraints, and the software-development in piece and performance. While the modification has been retaining, the structure of code has to be retested to gain a level of assurance that it will be conducted relying on the requirement functions and faults in the program. According to this situation, the test cases of each program must be generated in minimum numbers in order to improve the revised modules and reduce execution times through the entire software system, including saving cost of maintaining codes. Concepts of resolving this problem is ongoing by the techniques of regression test such as random and safe selection. The traditional techniques focus building methods for reduce the numbers of selected test cases, including the percent errors. Therefore, this article proposes the new technique that gives the lower numbers of test cases, while the faultless rate is higher than the traditional techniques.

Keyword: Software Maintenance, Random Selection, Regression Test Selection, Test Case Selection.

1. INTRODUCTION

Amounts of the new software versions are being created for various areas, e.g., business, education, natural science, including education and industry [1]. Therefore, the process of software maintenance is one of the most important following topics in software-development life cycle (SDLC) [2-3]. One of the major problems of software maintenance is to generate a suitable test pool (T) that is used to test the entire program before retesting the modified code [4]. Test pool includes a set of test cases (t) used for checking bugs, functions, and codes [4-5]. Particularly, if the size of the test pool is huge, then time consuming may reduce the abilities of the program [6]. Therefore, this paper proposes the techniques for selecting the test cases in order to speed up the testing time. Another issue after test case selection may refer to bugs, which is produced accidentally, during the maintenance. The reason is the reducing numbers of test cases may remove some essential elements, which should not be deleted from a test pool. In addition, they can affect the entire [7]. According to this if we remove the most test cases from the pool, and then we may lose the faultless rate after retesting the software [8]. In response to this, the regression test reduction techniques are proposed to solve the mentioned problems. The concepts of regression test selection, the size of a test pool must be reduced in order to choose the appropriate numbers of test cases, while the performance of the system is preserved, and especially the numbers of bugs should be solved [9]. The regression test minimization (RTM) involves removing irrelevant test cases. On the other hand, the regression test selection (RTS) chooses the appropriate test cases based on multiple regressions. Furthermore, the regression test prioritization (RTP) partitions the test cases into small groups and selecting the most relevant test cases. Obviously, those techniques can be applied for reducing the size of testing, but they may not maintain the new changes

* *Department of Information Technology Faculty of Science and Technology, Assumption University.*

after removal [10]. This paper concerns RS and RST as the comparative studies. The reason is they are often used by the programmers and testers [11]. The objectives of using those techniques depend on the requirements from users, stake holders, and managers. Relevant to the record, it shows that selecting all test cases in a test pool is simplest. However, it may introduce the problems of the cost of maintenance and time consuming. In the meantime, the random techniques can reduce the testing time, but it cannot stop the new bugs [9-12]. The safe technique can give the better results than others. Therefore, the technique of test case selection (TCS) is proposed to improve the abilities of reducing the numbers of test cases, including giving the higher faultless rate through the whole process of software maintenance. The challenge of TCS is to develop test case for the subject programs, a test pool, and the algorithm of selecting the representative due to the requirements. The factors in proposing the techniques are the number of functions, the lines of code, and the faulty versions [13]. In the parts of evaluation, the results show that TCS gives the lowest numbers of test cases, while the highest faultless rates are produced, whereas the comparative studies are also being applied.

2. THE CONCEPTS OF SOFTWARE MAINTENANCE

The process of software maintenance is shown in Fig. 1. There are seven steps are described as follows;

Step 1: Analyzing software; this step is required as the first step of the whole process of software maintenance. It is the analysis of the abilities of software to be modified in the step. It concerns the programming language, numbers of codes, and the changes that requested by users.

Step 2: Modifying software; it refers to the methods of adding, deleting, or modifying the previous source codes. This step needs the skills and experience of programmers.

Step 3: Building test pool; The test pool is produced as the test suite of the new software from some factors, eg., requirement specifications, line of codes, faulty versions.

Step 4: Generating test case; the numbers of test cases are generated and combined into a test pool.

Step 5: Selecting test case; in this step, there are many selection techniques are used. For examples, random, retest-all, control graph, data-flow, minimization, prioritization, and others are applied.

Step 6: Testing new software; this requires the knowledge of software testing such as V-model (verification and validation), unit module, system, satisfaction testing.

Step 7: Releasing the new software version; this step is denoted as the last step in the process of retaining software before deploying it to the users.

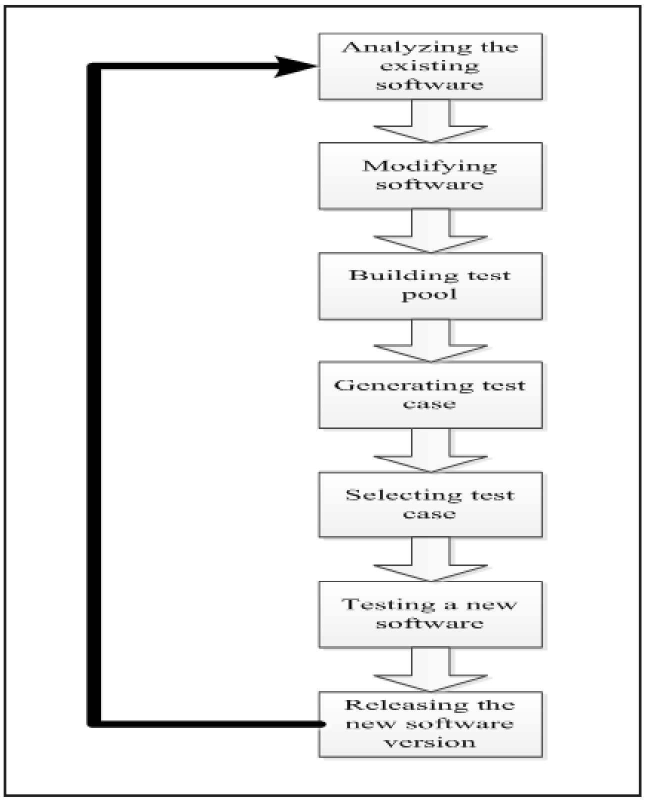

Figure 1. *The process of software maintenance.*

This paper focuses the techniques for selecting the test cases in Step 5 of the maintenance process. The oldest and simplest technique is the retest-all selection. It is the technique that simply uses all test cases in a pool for processing them in the next step. This technique is suitable when the size of a source code is proper or small. In order to measure the size, it depends on the developers' judgment. The problem starts when the size is getting bigger that causes running time increases. Unfortunately, that there are no reports about what size is called small or proper size. Another reason is about running time; it may refer to time consuming in searching data inside database or may be executing time for checking bugs in any lines of code [14]. That's why, this technique cannot response to faultless and time constraints, but no test selection tools are available, developers often select test suite based on "hunches", or loose associations of test suite with functionality, line of codes and faulty versions [15].

The Random Selection selects some number of test cases from test pool. The random algorithms can be varied by human judgments. This technique claims that it is a fast selection, which depends on the simplest functions. Particularly, the different numbers of random selection are required in one experiment. One of the majors studying with this technique is to observe, in which, what is the suitable random numbers that can reduce the maximum numbers of test cases. Besides this, it is also required to reduce the faults in a source code after running. However, we found that this technique cannot guarantee the abilities of reduction and faultless rate [16].

This paper focuses the regression test selection (RST) by Rothermel and Harrold because their results are better than retest-all, random selection, and others [16]. The RST constructs the control flow graphs for a program or procedure and its limited program and uses the flow graphs to select test cases that execute modified code from the original test suite. They describe that, under certain conditions, the set of test cases their technique selects includes every test case from the original test suite that can expose faults in the modified program or procedure. Particularly, although their algorithms may choose some test case that cannot expose faults, they are at least as accurate as other safe regression test selection techniques. Unlike many other regression test selection techniques, their algorithms can handle all types of program modifications and all language constructs. They have implemented their algorithms; initial empirical studies prove that their technique can significantly reduce the cost of regression testing modified program [17].

In this paper, the seven C programs, a number of modified versions, and the test pools for each program are material used in parts of experiments. Table I is the subject programs used in the part of experiments. These programs are originated by Hutchins and team. Particularly, they are written in C, and size varied from 138 to 516 lines of code. They applied a test pool of black-box to generate these programs which test cases using the category partition method with Siemens Test Specification Language tool (STSL)[18]. Afterward, the development team applied additional white-box technique to ensure that each exercisable statement, edge, and also definition-use for the base program or its control flow graph was exercised by at least 30 test cases. Hutchins and team also generated faulty versions of each program, which varied between 7 and 41 versions by modifying existing code in the base version; in most test cases, they provided a single line of code. In addition, a few cases are varied between 2 and 5 lines of code. The changes are realized either very easy to obtain. For example, there are more than 350 test cases in each test pool. However, it is very difficult to find the small numbers of test cases [19].

Table 1. *The subject programs.*

Name	F	L	B
print-tokens	18	402	7
print-tokens2	19	483	10
replace	21	516	32
schedule	18	299	9
schedule2	16	297	10
tcas	9	148	41
totinfo	7	346	23

3. METHODOLOGY

3.1 Produce Test Pool (T)

Test pool is the set of test cases. The sizes varied on functions *(F)*, line of codes *(L)*, and faults *(B)*. So,$T \propto F$ Relative to functions and codes: ; Bugs are probably found in the codes: ; Test pool varied by line codes: $T \propto C$.

$$T = c*L \quad or \quad c = \frac{T}{L} \tag{1}$$

When c is the expected value used for determining the test pool. Practically, in the experiment, each program is running at least 100 times to find c.

There are six steps to determine c described as follows;

Step 1: Generate the value of T for each subject programs.

Step 2: Estimate the value of L.

Step 3: Prepare 100 experiments.

Step 4: Run the experiments.

Step 5: Determine the value of c.

Step 6: Find the average of c.

The experiments are processed through the technique of using simulation to optimize the value of c.

The second consideration bases on $B \propto L$. The similar six steps are applied to find the value of c.

3.2 Select Test cases

In this section, the detail of the algorithm is created and used to determine the test cases in each test pool due to the subject programs. The algorithm of finding test cases is described as follows;

Step 1: Determine coverage area of T by T L

$$Cov(T) = \frac{T^2}{TxFxB} \tag{2}$$

Step2: Find coverage area of F by

$$Cov(F) = \frac{F^2}{FxTxB} \tag{3}$$

Step3: Find coverage area of F by

$$Cov(B) = \frac{B^2}{BxTxF} \tag{4}$$

Step4: Select test cases by

$$T = \sum (Cov(T)+Cov(F)+Cov(B)) \tag{5}$$

$$t = \frac{T^2}{TxFxB} + \frac{F^2}{FxTxB} + \frac{B^2}{BxTxF} \tag{6}$$

$$t = \frac{T^2 + F^2 + B^2}{TxFxB} \tag{7}$$

4. RESULTS AND DISCUSSIONS

4.1 Finding test pool

Due to the six steps of find the value c, the results from running 100 experiments for print-token, print-token2, replace, schedule, schedule2, tcas, and totinfo are 10, 8, 10, 9, 9, 11, and 3 respectively. According to these results, they are used in each subject program. Therefore, the results in Table 2 to 8 are the estimated T in the seven subject programs which can be produced relying to the values of c reported above. Moreover, Table 9 summarizes all records of T, including F, L, and B. The reasons that the results of finding T from Table II to VIII are not merged into one table are explained as follows;

Reason no. 1: The requirement specifications or functions of each program are different.

Reason no. 2: Faults or bugs are produced differently in the new programs.

Reason no. 3: Each c-value changes, although F is the same; this is because the complexities in each program are varied on L.

4.2 Selected test case

Table 10 shows the numbers of test cases selected by the proposed methods. According to this, there are 25 experiments conducted to find the total test cases used in the subject program, namely print_tokens. Due to the same algorithm of determining test cases, the results of print-tokens2, replace, schedule, schedule2, tcas, and totinfo are 105, 60, 114, 69, 78, 31, and 40 respectively. However, the numbers of selected test cases base on the mentioned factors, not including the complexity of source codes. Example of computation for print_token (see the results of row one as shown in Table 10):

$$t = \frac{T^2}{TxFxB} + \frac{F^2}{FxTxB} + \frac{B^2}{BxTxF}$$

$$= \frac{1531^2}{1531x2x3} + \frac{2^2}{2x1531x3} + \frac{3^2}{3x1531x2}$$

$$= 255$$

4.3 Comparative studies

In order to guarantee the quality of selecting test cases from each test pool of the subject programs by the proposed methods called TCS is better than some of the traditional techniques. The TCS will be compared with the Random Selection and Regression Test Selection, which is the well-known technique in the process of software maintenance and it is proposed by Rothermel and Harrold. However, those comparative studies are processed at least 20 experiments in order to avoid the bias of the selection.

4.4 Numbers of test cases

One of the most important criteria in measuring performance of the selection technique is to compare the amounts of chosen test cases for maintaining a new software version. The random technique and regression test selection are conducted at least 25 rounds as well as experimenting with the TCS. The results show that the numbers of test cases by the RS are higher than others. This explains that selecting test cases has no standard. According to this, in order to decide for the random technique depends on human judgments. The weakness of this technique is the bias of selecting cases, which relies on the frequency of testing. See all results in Table 11.

4.5 Ratio of selecting test cases in a test pool

The ratio of test case selection is used to evaluate the new software. The results describe amounts of test cases selected over the test pool. If the numbers are low, then the ratio gives low value as well. On other hands, if the test cases are large, then the ratio gets big. This explains that the cost of selection is expensive. See results of the comparative studies in Fig. 2.

4.6 Percent Reduction

The results of percent reduction among RS, RTS, and TCS are shown in Fig. 3. The graph of TCS is higher than the others. This means the abilities of selecting the minimum test cases by applying TCS is better than RS and RTS. Therefore, TCS can give the benefits to the maintenance process in terms of reducing testing and executing time, including the cost.

4.7 Percent faults

Another serious problem of maintaining software is the new bugs can be produced. Therefore, all of programmers tried to fix them to make the software run properly. The graph of percent faults after using those comparative studies as shown in Fig. 4. The graph shows that the percent faults are increasing until or bugs are fixed. Therefore, it begins constant whereas those bugs are disappeared.

The TCS technique can provide none of the bugs with the minimum numbers of test cases compared with both techniques. However, in the experiments, the faults are counted to be computed and used to compare the efficiency of the selection.

4.8 Suggestions

Up to now, on one prove what selection technique is the best. It depends on the setting hypothesis before doing the research. Therefore, the main factors that may fail the process of software maintenance are human judgment, skills, and experiences of programmers. Therefore, the selection techniques should be designed by covering those factors as possible.

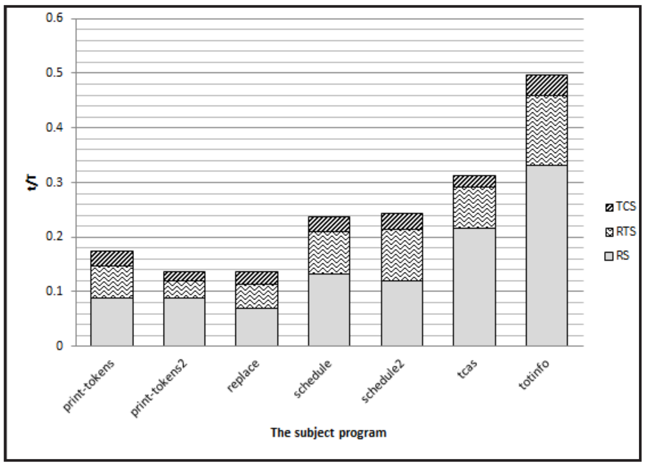

Figure 2. *The ratio of test cases per the pool.*

Table 2. *TEST POOL OF PRINT-TOKENS.*

No.	*F*	*B*	*c*	*T*
1	11	6	10	4020
2	16	5	9	3618
3	14	5	10	4020
4	17	5	9	3618
5	11	2	10	4020
6	12	7	9	3618
7	13	6	12	4824
8	12	6	10	4020
9	13	2	11	4422
10	15	3	11	4422
		average	**10**	**4060**

Table 3. *TEST POOL OF PRINT-TOKENS2*

No.	*F*	*B*	*c*	*T*
1	17	6	9	3618
2	10	5	7	2818
3	12	2	7	2814
4	10	7	8	3216
5	14	1	9	3618
6	12	1	9	3618
7	12	4	9	3618
8	18	8	7	2814
9	12	7	9	3618
10	12	5	8	3216
		average	**8**	**3961**

Table 4. *TEST POOL OF REPLACE*

No.	*F*	*B*	*c*	*T*
1	19	19	11	4422
2	17	25	9	3618
3	15	1	10	4020
4	20	7	10	4020
5	19	28	11	4022
6	18	30	11	4022
7	20	25	10	4020
8	21	18	11	4422
9	18	22	11	4422
10	19	15	9	3618
		average	**10**	**4975**

Table 5. *TEST POOL OF SCHEDULE*

No.	*F*	*B*	*c*	*T*
1	14	6	7	2093
2	15	8	9	2691
3	14	9	8	2392
4	14	9	9	2691
5	15	2	8	2392
6	14	8	10	2990
7	16	1	10	2990
8	14	9	9	2691
9	15	8	8	2392
10	15	3	10	2990
		average	**9**	**2631**

Table 6. *TEST POOL OF SCHEDULE2*

No.	*F*	*B*	*c*	*T*
1	15	2	6	1782
2	14	10	8	2376
3	16	1	10	2970
4	15	2	9	2673
5	16	7	9	2673
6	14	4	9	2673
7	15	2	9	2673
8	14	3	10	2970
9	15	9	10	2970
10	14	3	9	2673
		average	**9**	**2643**

Table 7. *TEST POOL OF TCAS*

No.	*F*	*B*	*c*	*T*
1	6	35	12	1776
2	8	2	10	1480
3	9	5	9	1332
4	8	1	10	1480
5	8	15	12	1776
6	7	12	10	1480
7	8	28	10	1480
8	8	9	11	1628
9	8	21	12	1776
10	7	13	10	1480
		average	**11**	**1569**

Table 8. *TEST POOL OF TOTINFO*

No.	*F*	*B*	*c*	*T*
1	7	7	3	1038
2	3	11	3	1038
3	5	22	3	1038
4	4	6	3	1038
5	5	23	4	1384
6	3	3	3	1038
7	6	20	2	692
8	3	21	3	1038
9	4	4	2	692
10	5	12	4	1384
		average	**3**	**1038**

Table 9. *THE SUMMARY OF THE TEST POOL IN EACHGRAM PROGRAM*

No.	*F*	*L*	*B*	*T*
print-tokens	18	402	7	4060
print-tokens2	19	483	10	3961
replace	21	516	32	4975
schedule	18	299	9	2631
schedule2	16	297	10	2643
tcas	9	148	41	1569
totinfo	7	346	23	1038

Table 10. *TEST POOL OF TCAS*

T	*F*	*B*	*T*
1531	2	3	255
1656	2	6	138
510	14	2	18
1784	5	1	357
3047	7	2	218
3381	1	6	564
460	7	6	11
3914	16	5	49
890	13	3	23
2446	15	6	27
934	4	1	234
2858	16	7	26
106	1	7	15
1659	5	7	47
234	3	3	26
3473	16	1	217
3927	18	4	55
2912	5	6	97
853	13	2	33
316	4	2	40
1407	17	7	12
1198	3	3	133
350	10	5	7
1045	4	5	52
3018	17	4	44
1996	10	5	40
		average	**105**

Table 11. *THE SUMMARY OF TEST CASES IN THE SUBJECT PROGRAMS*

No.	*T*	*t*
print-tokens	4060	105
print-tokens2	3961	60
replace	4975	114
schedule	2631	69
schedule2	2643	78
tcas	1569	31
totinfo	1038	40

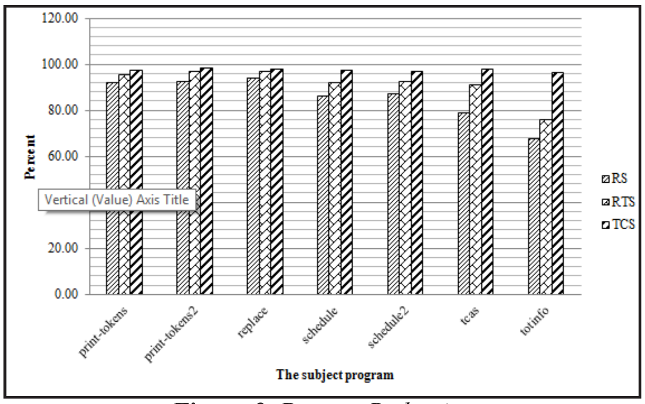

Figure 3. *Percent Reduction.*

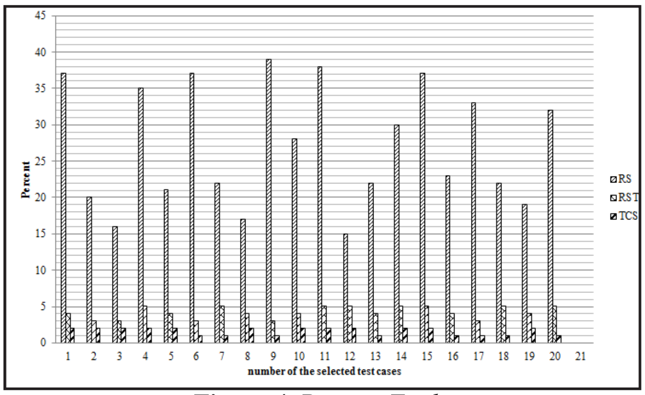

Figure 4. *Percent Faults.*

5. CONCLUSIONS

This research article presents two contributions, which are the higher recent reduction and faultless when compared with the random and regression test selection techniques. However, the TCS cannot guarantee that it is the best technique because there are many complexities in the development software. For example, they involve the constraint in the process of maintenance (e.g., functions, faulty version, bugs, requirement changes, software and hardware configuration, and others). Those factors may decrease the abilities of the entire source code while retesting, rerunning and re-debugging the programs, particularly, in all processes of maintaining software mostly spent very long time. By two main objectives, the selected test cases must not affect the performance of keeping faultless after the whole process is obtained. In the future works, we should look for the maintenance methods that can be used to cover many more numbers of factors to build the better test case selection, eg., applying the concepts of case-based maintenance. It is one of the AI systems that can produce a good test case to represent the situation that can be occurred during the development process. Furthermore, the deletion and addition techniques provided for the area of case-based maintenance are applicable for various fields.

6. REFERENCES

[1] A. lawanna. “The Theory of Software Testing.” *Australian Journal of Technology*, Vol. 16, No. 1, pp. 35-40, 2012.

[2] A. Alain. “Studying Supply and Demand of Software Maintenance and Evolution Services.” *In Proceedings of the 7th International Conference on the Quality of Information and Communications Technology*, *IEEE Computer Society*, Porto, USA, pp. 352-357, 2010.

[3] A. M. Davis, H. Bersoff and E. R. Comer. “A Strategy for Comparing Alternative Software Development Life Cycle Models.” *Journal IEEE Transactions on Software Engineering*, Vol. 14, No. 10, pp. 1462-1477, 1988.

[4] S. Bjorklund, M. Pribytkova and T. Karaulova. “Development the Maintenance Plan: Maintenance Activities on Operational Level.” *In Proceedings of the 7th International Baltic Conference Industrial Engineering*, pp. 1-6, 2010.

[5] J. Barton, E. Czeck, Z. Segall and D. Siewiorek. “Fault injection experiments using FIAT.” *IEEE Transactions on Computers*, Vol. 39, No. 4, pp. 575-582, 1990.

[6] E. Jenn, J. Arlat, M. Rimen, J. Ohlsson and J. Karlsson. “Fault injection into VHDL models: The MEFISTO tool.” *In Proceedings of the 24th IEEE International Symposium on Fault Tolerant Computing*, pp. 66-75, 1994.

[7] H. K. N. Leung and L. J. White. “Insights into Testing and Regression Testing Global Variables.” *Journal of Software Maintenance: Research and Practice*, Vol. 2, No. 4, pp. 209-222, 1990.

[8] H. Agrawal, J. Horgan, E. Krauser and S. London. “Incremental regression testing.” *In Proceedings of the Conference on Software Maintenance*, pp. 348-357, 1993.

[9] M. V. Zelkowitz, D. R. Wallace and D. W. Binkley. “Experimental Validation of New Software Technology.” *Lecture Notes on Empirical Software Engineeing*, pp. 229-263, 2003.

[10] H. Agrawal, J. Horgan, E. Krauser and S. London. “Incremental Regression Testing.” *In Proceedings of Conference on Software Maintenance*, pp. 348-357, 1993.

[11] V. R. Basili and R. W. Selby. “Comparing the Effectiveness of Software Testing Strategies.” *IEEE Transaction on Software Engineering*, Vol. 13, No. 2, pp. 1278-1296, 1987.

[12] E. Wong and A. P. Mathur. “Fault Detection Effectiveness of Mutation and Data-flow Testing.” *Software Quality Journal,* Vol. 4, No. 1, pp. 69-83, 1995.

[13] G. Rothermel and M. Harrold. “A safe efficient regression test selection technique.” *ACM Transaction on Software Engineering*, Vol. 6, No. 2, pp. 173-210, 1997.

[14] G. Rothermel and M. Harrold. “Empirical studies of a safe regression test selection technique.” *IEEE Transaction on Software Engineering*, Vol. 24, No. 6, pp. 401-419, 1998.

[15] F. I. Vokolos and P. G. Frankl. “Empirical evaluation of the textual differencing regression testing technique.” *In Proceedings of the International Conference on*

Software Maintenance, pp. 44-53, 1998.

[16] G. Rothermel and M. Harrold. “Analyzing regression test selection techniques.” *IEEE Transaction on Software Engineering*, Vol. 22, No. 8, pp. 529-551, 1996.

[17] G. Rothermel and M. Harrold. “A safe efficient regression test selection technique.” *ACM Transaction on Software Engineering*, Vol. 6, No. 2, pp. 173-210, 1997.

[18] G. Rothermel and M. Harrold. “Empirical studies of a safe regression test selection technique.” *IEEE Transaction on Software Engineering*, Vol. 24, No. 6, pp. 401-419, 1998.

[19] M. Hutchins, H. Foster, T. Goradia and T. Ostrand. “Experiments on the effectiveness of dataflow-and control flow-based test adequacy criteria.” *In Proceedings of the 16th International Conference on Software Engineering* (ICSE’94, Sorrento, Italy, May 16-21), B. Fadini, L. Osterweil, and A. V. Lamsweerde. Chairs. *IEEE Computer Society Press*, Los Alamitos, CA, pp. 191-200, 1994.

[20] T. Graves, M.J. Harrold, J. M. Kim, A. Porter and G. Rothermel. “An empirical study of regression test selection techniques.” *In Proceedings of the 20th the International Conference on Software Engineering*, pp. 188-197, 1998.