# Using Program Slicing Technique to Reduce the Cost of Software Testing

Asghar Mohammadian [1], Bahman Arasteh [2]
[1]Department of Computer Engineering, Ilkhchi Branch, Islamic Azad University, Ilkhchi, Iran.
Email: asgharmohammadian@gmail.com
[2]Department of Computer Engineering, Tabriz Branch, Islamic Azad University, Tabriz, Iran

## ABSTRACT
*Systems of computers and their application in the lives of modern human beings are vastly expanding. In any kind of computer application, failure in computer systems can lead to a range of financial and mortal losses. Indeed, the major origin of software failure can be located in designing or implementing software. With regard to these statistics, 30% of the software projects have been prosperous and successful. The proposed method is intended to reduce the cost and time of testing and it focuses on enhancing the efficiency of software testing methods. In this paper, we investigated the effect of slicing techniques on the reduction rate of testing cost and time. The results of experiments show that we can cover a large number of program instructions, branches and paths by a small number of test cases in the sliced program.*

**KEYWORDS:** Software testing, Cost, Program slicing, Coverage

## 1. INTRODUCTION

Systems of computers and their application in the lives of modern human beings are vastly expanding. The citizens of the global village, witness new aspects and facets of computer systems in their routine lives. Hence, information technology and software systems are increasingly involved in our lives. Various industrial, military applications of computers, applications in communications and transportations are evidences of their use in our lives.

In any kind of computer application, failure in computer systems can lead to a range of financial and mortal losses. As a case in point, in the new millennium, computers play significant roles in a vast range of applications from air traffic navigation systems, nuclear reactors, aircraft systems, sensor networks, industrial processing systems and medical instruments. Hence, the computer systems which are used in such important and critical systems should have remarkable dependability and reliability. The capability of a software system can be defined in terms of dependable and reliable services and applications.

The results of the conducted experiments in the computer and software literature indicate that a 25-percent increase in the complexity of issues results in a 100-percent complexity of the related programming and the number of program lines. For instance, the number of code lines of a software used inside a modern automobile is approximately 19000 lines and in the navigation software of airplane systems a few hundred thousand lines of

codes might be used. Error occurrence in the software systems of critical applications might lead to un-returnable and un-rectifiable losses [1, 16, 17].

Indeed, the major origin of software failure can be located in designing or implementing software. Table 1 illustrates the percentage of successful and unsuccessful software projects in the year 1996 and Table 2 shows the same statistics on the status of software projects in the year 2000 [1, 8, 9 ,10, 11, 16]. With regard to these statistics, 30% of the software projects have been prosperous and successful [8, 10, 11, 16].

It should be noted that the issues and concepts which are related to software error should be considered in the software production phase. The financial and mortal losses resulting from software failure motivates and forces the software producers to sort out and resolve the errors in the process of producing software and after the code is produced, they try and experiment the software to locate and resolve the errors. Thus, it can be argued that spotting out the errors and enhancing the dependability and reliability of software is of high significance.

**Table 1.** The percentage of successful and unsuccessful software projects in the year 1996

| Software Projects Status 1996 | | |
|---|---|---|
| Cancelled | Challenged | Successful |
| 31% | 53% | 16% |

**Table 2.** The percentage of successful and unsuccessful software projects in the year 2000

| Software Projects Status 2000 | | |
|---|---|---|
| Cancelled | Erroneous | Successful |
| 28% | 49% | 28% |

## 2. THE OBJECTIVES OF THE STUDY

The objective of conducting this research study is to propose methods for testing and evaluating application software. The proposed method is intended to reduce the cost and time of testing and it focuses on enhancing the efficiency of software testing methods. As the results of the related studies have revealed, the different parts and components of a program have effects on the amount and quality of the software output. Hence, in the proposed method, the inputting program will be first analyzed and those parts which impact the results will be identified; as a result, only some parts of the program code for carrying out operations will be selected. Thus, testing only the effective arts of the program will reduce the required time and cost. An important issue which should be considered is that the program should be both statically and dynamically analyzed in order to identify the effective parts of the program and conduct technical tests.

Inasmuch as the code sizes are usually massive and overwhelming and there are complex relationships among program commands, dynamic analysis of the program is considered to be complicated and difficult. The method proposed in this study can help improve the following variables:

- Saving the required time for the testing process
- Saving the required costs for the testing process

## 3. REVIEW OF THE RELATED LITERATURE

The size of data in an organization can be an index of the complexity of software systems and applications. Data size has rapidly increased during the past forty years [1]. Software quality is a software feature which refers to the consistency and compatibility of the software with the operational needs; also, it can also be defined as the well-defined efficiency of the software and should be in line with standards of development [2]. Many instances of software failure can be found where the failure leads to critical and undesirable conditions and situations. Software is regarded as the unobservable part of a system and this invisibility of the software enhances its importance in terms of reliability. Software reliability can be considered as a process of executing a program to find its errors [2]. The following issues are considered in software testing:

- What aspects of software are considered and analyzed in software testing?
- How much time is required for software testing?
- How much does it cost to test and evaluate a software?

It is assumed that using a software test which covers and analyzes most parts of the software code can locate and find more of the software errors; as a result, software testers can find more accurate results and findings about software reliability. However, with regard to the input intervals of a software, it should be argued that implementing a comprehensive test even on a software including only 100 code lines calls for overly vast amounts of time and cost. In many recent studies, this issue has been regarded as a research challenge for researchers. Testing those softwares which are related to safety-critical applications might cost three, four or five times as much money as the entire production cost of other softwares. The complete and comprehensive testing of the different aspects of a software can be seldom realized.

Some researchers contend that they should reduce and limit the number of tested features and aspects so that the test cost can be reduced [8]. The main drawback of this testing method is that the degree of covered errors in this method will be limited and hence the number of discovered errors will be significantly smaller. The studies [5, 7, 9] used some methods in which program controlling graphs were used to produce test features. These studies highlighted the significance of node coverage and edge coverage. Using control graph and its coverage as a test criterion facilitates the production of test data. Also, this method can be used for testing the designing stages of a software.

The method proposed in [6] covers and tests the conditional commands and branches of a program. However, the main shortcoming of this method is that the errors related to the other blocks of the program (non-conditional commands) cannot be discovered. The method proposed in [10, 11] reduces the testing time, so that the time and cost involved in testing can be controlled and reduced. In this method, an attempt is made so that the degree of error coverage would not be reduced.

The studies reported in [12, 13] used discovery methods to minimize the time and cost of testing. Using discovery methods to produce test data will lessen the time and

cost of software testing. The method proposed in [14] evaluates the degree of test efficacy. In this paper, error coverage rate and the required time have also been considered. The study reported in [15] considers coverage criterion as a criterion for examining the testing techniques.

With regard to the above-mentioned studies, it can be maintained that coverage rate, testing time and cost have been examined and considered as the variables and testing criteria in all of the software testing methods. In the method proposed in this study, the magnitude of the effectiveness of the different blocks and sections on the program output are analyzed so that the different blocks and functions of a software can be classified. In the method which has been offered and examined in this paper, those parts of the program which have no impact on the program output are identified in a two-phase process and hence they will be excluded from testing. Indeed, the testing process will be implemented on the sensitive and effective parts of the program. This method will determine the degree of the effectiveness of the different parts on the program output and then it will focus on and emphasize covering and analyzing the vulnerable aspects of the program. In other words, according to the proposed method, covering the sensitive and effective commands and data, rather than covering all commands and data indiscriminately will result in time reducing and controlling the time and cost which are involved in the program testing process.

## 4. PROPOSED METHOD

As mentioned in the literature review, much time consumption, high cost and low efficiency are regarded as generic shortcomings of the commonly used methods for software testing. Hence, developing efficient, low-costing and automatic methods of software testing is considered to be a research challenge of this area. In the present study, an attempt was made to develop an efficient, fast and low-costing method and the method was designed to have as high coverage as possible for application software. To minimize and lessen the testing cost, the proposed method analyzes the input portions of the program and hence finds those sections which have an effect on the program output. As a result, the testing operation will be carried out on only those effective section which have been identified.

In line with the purpose of the study, the proposed method focuses on those arts of the program which have significant impact on its output. Those parts of the program code which are not directly involved in program output are eliminated from the testing process. As regards the identification of those sections which influence the program output, both of the following methods will be used:

- Static analysis of the program
- Dynamic analysis of the program

In the first phase, the code of the input program will be statistically examined and analyzed. In this phase, those parts of the code which have no effect on the program output should be statically and syntactically identified.

Accordingly, the parts and code which are related to the qualitative needs of the program will be removed. Hence, as the size of the code is reduced in this way, testing

time and cost will also decrease. However, since the first phase of the program was done statically and analytically, only some of the ineffectual parts of the program could be identified. Thus, it can be argued that after the static analysis of the program, a significant amount of ineffectual still remains in the program. Therefore, the second phase of the proposed method makes use of the dynamic technique identifies the remaining parts of the code which have no effect on the output.

The purpose doing the second phase is to dynamically examine the different parts of the program code and spot out those parts which were left unidentified. When compared with static analysis, the dynamic approach can locate significantly more ineffective parts of the code. It should be noted that the reason for using the static analysis before the dynamic analysis is that the dynamic analysis requires much time due to the need for running the program; hence, using the static analysis before the dynamic analysis can save some time. Figure 1 below represents the schemata of the proposed method.

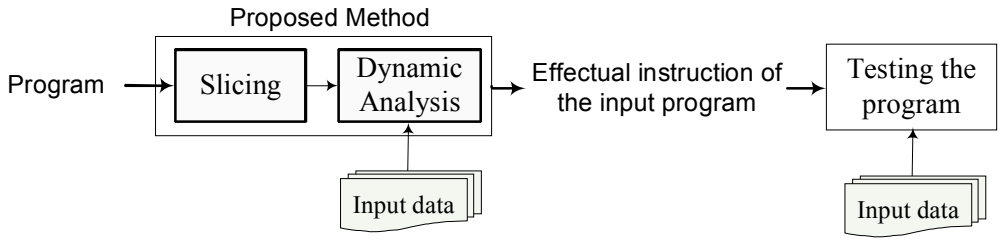



**Figure.1.** Schemata and blueprint of the proposed method for reducing test cost

Indeed, it should be mentioned that only those commands which have an impact on program output are kept in the program and the other commands are eliminated. Therefore, in the second phase of the method which focuses on the dynamic behavior of the program, some other parts of the program will be removed.

Finally, throughout the two phases of the method, the ineffectual parts of the program will be eliminated and the result will be a simplified version of the program. The simplified version is equal to the main program. Thus, the time and cost involved with the simplified version will be in turn less than those of the main program.

Slicing a technique is used to identify program commands which calculate a series variables at a certain point of the program. Slices are produced based on the slicing criterion. The slicing criterion is in turn determined based on the arranged pair (v, n). V refers to a series of variables and n is a node in the control graph of the program. In the program slicing, graph nodes of a program such as p which are not able to change the value of V variables are removed so that a slice p can be produced with regard to (v, n). Figure 2 shows an original program and Figures 3 and 4 show its slices based on different variable.

```
Algorithm find_max_min()
```

```
1)  {
2)  b = a[0];
3)  s = a[0];
4)  for (i =0; i<=max ; i++)
5)    {
6)      if( b < a[i])
7)                b= a[i];
8)      if( s > a[i])
9)                s =a[i];
10)    }
11)   printf("%d", b);
12)   printf("%d", s);
13) }
```

**Figure. 2.** The source code of a program for finding the max and min of a list

As the Figure 2, 3 and 4 illustrate, the program calculates the values of the highest and lowest elements of an array and saves the results in b and s; with regard to the output, the related slices include the main parts of the program and the related calculations.

```
Algorithm find_max_min()
1)  {
2)   b = a[0];
3)   for (i=0;i<=max ; i++)
4)    {
5)      if( b < a[i])
6)         b= a[i];
7)     }
8)    printf("%d", b);
9)  }
```

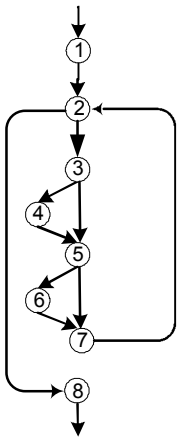
**Figure.3.** Sliced version of program in Fig. 2 based on variable *b*

```
Algorithm find_max_min()
1) {
2)   s = a[0];
3)   for (i =0; i<=max ; i++)
4)    {
5)      if( s > a[i])
6)         s =a[i];
7)     }
8)    printf("%d", s);
9)  }
```

**Figure.4.** Sliced version of program in Fig 2 based on variable *s*

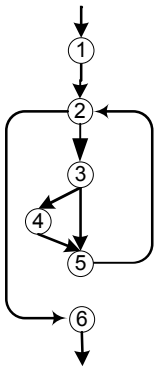
The program lines are numbered to recognize the program nodes and the used slices. These figures illustrate the effect of slicing on determining the effective calculations of the output. Static backward slicing has been used in this figure [18].

Using the program slice rather than the main program reduces the degree of program complexity. Hence, with regard to the available commands and data in a program slice, the number of tests will be accordingly reduced. As a case in point, the control flow graph of the programs in Figure 2 and Figure 3 shown in Figure 5. If node coverage is considered as the test requirement, then we need three test data (2, 1, 3) for covering all nodes in the original program. Indeed, these three test-data lead to execute all instructions in the original program. Whereas original program needs three test-data, the sliced program needs only two test-data (1, 2) to cover all instruction. As a result, the number of test-data for testing the sliced program is lower than the number of test-data for the




a) CFG of the original program

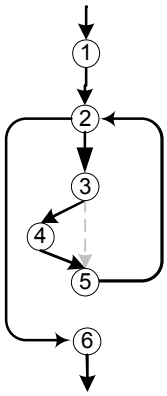
TR[1]: {1, 2, 3, 4, 5, 6, 7, 8}
Test case: a[]={2, 1, 3}



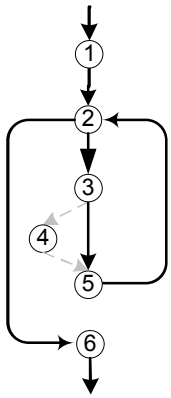

b) CFG of the sliced program

TR: {1, 2, 3, 4, 5, 6,}
Test case: a[]={1, 2}

**Figure.5**. The control-flow graphs for the original and sliced version and the corresponding test cases for covering instructions




a) Executable $path_1$ which covers {(2,3),(3,4),(5,2),(2,6)}




b) Executable $path_2$ which covers {(2,3),(3,5),(5, 2),(2,6)}

TR = {(2,3) , (2,6), (3,4), (3,5)}
Test case: a[] = {۲و۱}

**Figure. 6.** The paths in the sliced program and the corresponding test cases for branch coverage

[1] Test Requirements

corresponding original program. Testing a program with a small number of test data will decrease the time and cost of software testing. Testing sliced program rather than the original program will decrease the cost and time of software testing.

Figure 6 (a) illustrates the required test cases for the branch coverage in the original program and sliced program. Regarding the control-flow graph of the original program, the input array should be initialized with {2, 1, 3}. Figure 6 (b) illustrates the designed test case for covering the branches of the sliced program. If the input array is initialized with the {2, 1}, all the branches in the sliced program will be covered. Thus, the number of required test case for covering branches in the sliced program is lower than the number of required test case for the original program. As a result, testing a sliced software have lower cost and time than the original software.

## 4. EXPERIMENTS AND RESULTS

This section presents the results of experiments that have been conducted to evaluate the effectiveness of the proposed method in the reduction percentage of testing cost. A set of four different programs were used in the experiments. We developed two different versions of each program. The original program is the first version and the slice of the same program using the Amorphous Slicing technique is the second version. The following programs, all written in the C programming language, have been employed in the experiments:

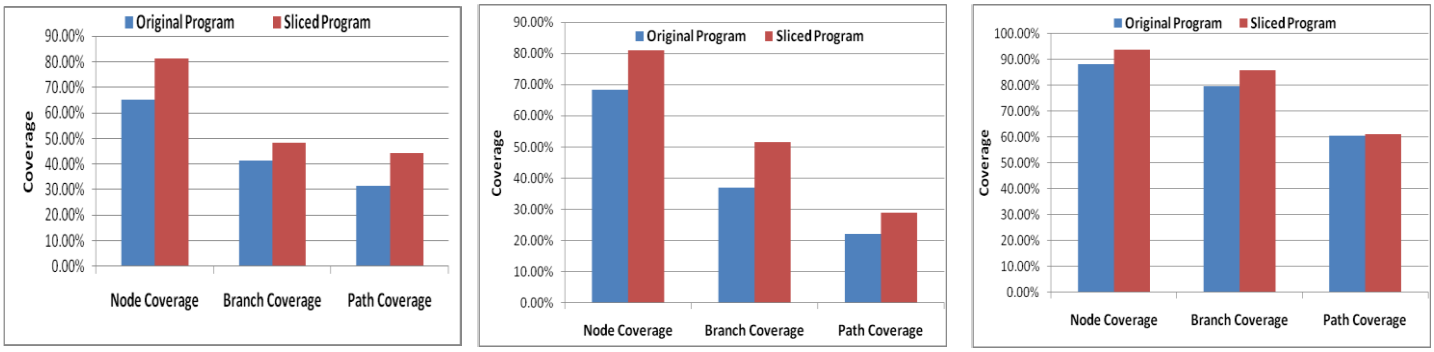
- Matrix multiplication (MM): a classic matrix multiplication algorithm is used to multiply to 10*10 matrixes.
- Bubble sort (BS): an efficient version of the bubble sort algorithm is used to sort a list of 100 integer elements.
- Binomial Coefficient (BC): a bottom-up dynamic algorithm is used to compute the binomial coefficient for value *n* and *k*.

These programs were compiled by the GCC compiler. The node, branch and path coverage are taken into consideration in the experiments. The required test data has been generated randomly with unified distribution. Indeed, a same test-set was used for testing original and sliced programs. Regarding the results of experiments, using the same number of test data in the sliced program have higher node, branch and also path coverage. Attaining 100% coverage in the slices program needs a lower number of test data and consequently lower time than the original program. Figure 7 illustrates the effect of slicing technique on the reduction of the number of test-cases and consequently on the reduction of cost and time.

In our experiments, the behavior of the main (original) program is equal to that of the program slice. Therefore, running the testing process will on the slice will require a lower number of test data. In case one can cover large parts of the program by fewer numbers of test cases and test data, the testing cost will hence be reduced. This is done here via the slicing technique.

## 5. CONCLUSIONS

In this paper, we investigated the effect of slicing techniques on the cost and time of software testing. The results of experiments show that we can cover a large number of program instructions, branches and paths by a small number of test cases in the sliced program. Indeed, using slicing techniques in the software-testing process will decrease the number of required test cases and consequently the cost and time of testing will be decreased.




a)Testing MM program with 20 randomly generated data

b)Testing BS program with 15 randomly generated data

c)Testing BC program with 20 randomly generated data

**Figure.7.** The amount of node, branch and path coverage in the original and sliced program with the same test cases

## REFERENCES


[1] M.R. Lyu, "*Handbooks of Software Reliability Engineering"*, ISBN 0 -07-039400-8, IEEE-CS Press, 1995.

[2] R.S. Pressman & Associates, "Software Engineering: A Practitioner's Approach, 6/e", copyright 2005.

[3] T. Adams, "*Total Variance Approach to Software Reliability Estimation*", IEEE Trans.SoftwareEngineering,Vol.22, No.9,1996,pp.687-688.

[4] P. Popic, D. Desovski, W. Abdelmoez, and B. Cukie, "*Error Propagation in the Reliability Analysis of Component Based Systems*," Intel Symposium. Software Reliability Eng.,pp.53-62,2005.

[5] B. Beizer, "*Software Testing Techniques*", Second Edition. New York: Van Nostrand Reinhold, 1990.

[6] A. Bertolino and M. Marre´, "*How Many Paths Are Needed for Branch Testing*?" The J. Systems and Software, vol. 35, no. 2, pp. 95-106, Nov. 1996.

[7] P.G. Frankl and E.J. Weyuker, "*An Applicable Family of Data Flow Testing Criteria*" IEEE Trans. Software Eng., vol. 14, no. 10, pp. 1483-1498, Oct. 1988.

[8] M. Marre and A. Bertolino, "*Reducing and Estimating the Cost of Test Coverage Criteria*," Proc. 18th Int'l Conf. Software Eng. (ICSE 18), pp. 486-494, Mar. 1996.

[9] S. Rapps and E.J. Weyuker, "*Selecting Software Test Data Using Data Flow Information*," IEEE Trans. Software Eng., vol. 11, no. 4, pp. 367-375, Apr. 1985.

[10] G. Rothermel, M.J. Harrold, J. Ostrin, and C. Hong, "*Empirical Study of the Effects of Minimization on the Fault Detection Capabilities of Test Suites*," Int'l Conf. Software Maintenance, pp. 34-43, Nov. 1998.

[11] W.E. Wong, J.R. Horgan, S. London, and A.P. Mathur, "*Effect of Test Set Minimization on Fault Detection Effectiveness*," Proc. 17th Int'l

Conf. Software Eng. (ICSE 17), pp. 41-50, Apr. 1995.

[12] C. R. Reeves, "Modern Heuristic Techniques for Combinatorial Problems", McGraw-Hill, 1995.

[13] D. Corne, M. Dorigo, and F. Glover, editors. "*New Ideas in Optimization*", McGraw-Hill, 1999.

[14] A. Mockus, "*Test Coverage and Post-Verification Defects: A Multiple Case Study*", In the 3rd International Symposium on Empirical Software Engineering and Measurement, ESEM 2009.

[15] Y. Wei, "*Is Coverage a Good Measure of Testing Effectiveness*?", Chair of software engineering ETH Zurich, CH-8092 Zurich, Switzerland.

[16] H. Pham, "*System Software Reliability*", Springer Series in Reliability Engineering, ISBN 0 -07-039400-8, Springer-Verlag London Limited 2006.

[17] P. Ammann, J. Offutt, "Introduction to Software Testing", CAMBRIDGE UNIVERSITY PRESS, ISBN 978-0-521-88038-1, 2008.

[18] 18 .G. A. Venkatesh, "*The semantic approach to program slicing*", In ACM SIG-PLAN Conference on Programming Language Design and Implementation, pages 26, 1991.

[19] B. Korel, J. Rilling, "*Dynamic program slicing methods*", Information and Software Technology special Issue on Program Slicing, volume 40, pages 647, 1998.

[20] G. Canfora, A. Cimitile, "*Conditioned program slicing*", Information and Software Technology special Issue on Program Slicing, volume 40, pages 595, 1998.

[21] S. Horwitz, T. Reps, D. Binkley, "*Inter-procedural slicing using dependence graphs*", ACM Transactions on Programming Languages and Systems, 1990.