I. J. Computer Network and Information Security, **2017, 5, 21-30**
Published Online May 2017 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijcnis.2017.05.03

Enhancing Software Reliability against Soft-Error using Minimum Redundancy on Critical Data

Saeid A. Keshtgar
Department of Computer Engineering, Tabriz Branch, Islamic Azad University,
Tabriz, Iran
E-mail: saultimate@gmail.com

Bahman B. Arasteh
Department of Computer Engineering, Tabriz Branch, Islamic Azad University,
Tabriz, Iran
E-mail: b_arasteh@iaut.ac.ir

Abstract—Nowadays, software systems play remarkable roles in human life and software has become an indispensable aspect of modern society. Hence, regarding the high significance of software, establishing and maintaining software reliability is considered to be an essential issue so that error occurrence, failure and disaster can be prevented. Thus, the magnitude of errors in a program should be detected and identified and software reliability should be measured and investigated so as to prevent the spread of error. In line with this purpose, different methods have been proposed in the literature on software reliability; however, the majority of the proposed methods are inefficient and undesirable due to their high overhead, vulnerability, excessive redundancy and high data replication. The method introduced in this paper identifies vulnerable data of the program and uses class diagram and the proposed formula. Also, by applying minimum redundancy and duplication on 70% of the critical data of the program, the proposed method protects the program data. The evaluation of the operation of the propose method on program indicated that it can improve reliability, reduce efficiency overhead, redundancy and complexity.

Index Terms—Reliability, redundancy, failure, fault, error, performance overhead.

I. Introduction

Computer-based systems are ubiquitous in all different areas of modern life from house appliances such as microwave ovens and washing machines to complex application programs like airplanes, trains, medical control systems, etc. The costs and consequences of the failure of these systems can be so disastrous and catastrophic that serious injuries, harms and life losses can result from them. Such failures can devastate computer system, breach security, lead to the collapse of business or the loss of opportunities. Indeed, such misadventures and non-successes are related to defects which interrupts systems and cause incompatibilities among them. Your goal is to simulate the usual appearance of papers in a Journal of the Academy Publisher. We are requesting that you follow these guidelines as closely as possible. One fault or defect can cause an error or failure. It is sometimes referred to as a bug. Indeed, a fault is an abnormal condition which occurs in the software or hardware of a system [22]. One of the important challenges in designing computer systems is soft error. Also, soft errors which are due to radiation are regarded as a key challenge in designing computer systems [32].

As the complexity increases and the demand for the required quality in the markets goes up, the need for designing reliable digital systems has become increasingly essential [26]. For achieving high reliability and delivering it to customers, two major attempts should be made. First, architects should figure out the impact of soft errors in their designing. Second, they should make a smart choice among the available methods for reducing the impact of soft errors so that maximum reliability and minimum overhead can be achieved [28].There are two basic methods for establishing reliability; the first one is to avoid error by using specifications and acknowledging formal methods and the second method is a highly meticulous and precise process for developing software. Hence, the enhancement of software reliability by reducing redundancy and performance overhead is another notable issue in this research domain [21].

From a system point of view, two highly critical features are software quality and reliability [19]. Reliability can be regarded as a remarkable criterion or standard for measuring the quality of a system [14].Since software can be examined and observed as an important component of a system, system analysts investigate software reliability as an indication and signal of the entire system [4].Software reliability enables a program to tolerate and resist against the probable errors which might occur. Indeed, acceptable software reliability can enable the program to properly function within a specific period of time [10]. Furthermore, good reliability guarantees that the software operate at a certain level in spite of the presence of several faults and errors so that

the system does not fail. It should be noted that for achieving high reliability for a software, the number of errors should be reduced. For reducing the number of errors, the three factors of error prevention, error detection and error tolerance should be used [31].

Regarding software reliability, several research studies have been conducted and many methods have been proposed. However, some of the proposed methods have not been effective yet. Software methods which are used for tolerating soft errors and enhancing reliability may result in a significant increase of volume and execution time. Hence, it should be underscored that reducing performance overhead in the available software methods

is number one priority. Performance overhead refers to the slowing down of program execution and the increasing of the volume of program instructions and commands. Performance overhead in real-time systems can lead to intolerable delay and finally system failure. The notable issue regarding performance overhead is that it increases system complexity. Consequently, system complexity can cause irreparable and irrecoverable costs. The reduction of each of the above-mentioned factors can optimize and improve software reliability. Although software cannot be seen or touched, it should be necessarily used in computer systems so that they can fulfill the intended functions and applications. Thus, it can be maintained that computers have become a vital component of the modern society [33].Hence, in general, software-based systems are aimed at satisfying system users ;for satisfying system users in this research study, two main objectives were taken into consideration: the first objective was to enhance software reliability via applying minimum software redundancy on critical data of the program and the second objective was to reduce performance overhead caused by the imposed redundancy.

The paper is organized in this way: after the introduction section which was discussed above, the research method is described in the following section. Then, the related works are briefly overviewed. Next, the details on the experimental environment are given. After that, the results and discussion of the results are mentioned. Finally, the conclusion to the study and some suggestions for further research are given.

II. Related Works

In general, it can be pointed out that the continuity of services can result in reliability [8]. For achieving high reliability, errors and mistakes should be reduced and eliminated. Also, vulnerable data of programs should be identified so that they should be made resistant against probable errors. In the followings, some of the recent research studies conducted in this filed are overviewed.

Recently, several studies with regard to embedding strategic detectors in program code have been carried out. Hiller et al [25] used error propagation analysis (EPA). It is assumed that ideal coverage (100%) has been investigated and signals are located in spots where the probability of error detection is practically high. The avalanche paradigm of Voas [17] is regarded as a method for mentioning statements before faults in the program propagate which has been proposed for the critical modes. Goradia [7] examined the sensitivity of the values of the erroneous data in terms of software testing. DAIKON[24] used a dynamic analysis system for the probable production of the features of the program for software faults.

Narayanan et al, [29] used the features of the loops produced by DAIKON for detecting soft errors in data cache. DAIKON's claims are about locations at the beginning and end of loops and the recall methods. However, providing error detection with little delay time may not be sufficient. For example, systemic/application programs might have long improper performances before they reach the acknowledgement spot. Benso et al [23] proposed a compiler method for detecting critical values in a program. Indeed, the sensitivity of a variable is measured based on its life span and the way in which other variables influence it. This method can protect data against errors which have been derived from a critical variable and propagate to other variables. However, this method cannot protect against errors which have been propagated from other locations to the critical variable.

Pattabiraman et al [30] investigated detecting placement in programs for fault detection and the cases which are attributed to errors in data values. Their objective was to detect low potential and prevent error propagation. The criteria for strategic production from ideal detectors in the locations of programs are introduced based on the measured criteria. Measurements were implemented in the form of a dynamic dependent graph. Directional graph is non-rotational which produces a train of dynamic dependencies among values which have been created during a round of program execution. The coverage of detectors is investigated by means of injecting errors in real programs. The results indicated that few number of the embedded strategic detectors can achieve a high degree of coverage.

The majority of research studies show that, in computer-based systems, a high percentage of errors lead to silent data corruption [11] [12].That is to say, the system might produce inaccurate results though it seems that the program has finished properly. This behavior is mainly produced by the faults of the pure data. For example, faults and errors appear in spots of data storage memory or in the microprocessor registers. Hence, such a computer-based system is called to be silently corrupted although the outputs have proper results, i.e. the system does not produce inaccurate outputs.

In [13], the researchers found that for achieving a high degree of soft corruption in average computers with error detection mechanisms, a set of precise software methods for error detection has been selected. These methods include ABFT (algorithm-based fault tolerance), expressions, time redundancy and checking flow control [3] [15] [5] [6] [1]. ABFT is a highly effective method but it lacks generality. It is useful for application programs with regular structures. However, it can be used for a limited number of problems [13].

Using logical statements in different spots included in

program for reversing the fixed relations among the program variables can lead to different problems because statements for the programmer and its efficacy on the program and programmer skills are not clear. In contrast, methods based on time redundancy focus on time exploitation in idle times while executing program for duplicating measurements and comparing error detection [2] [9].

The initial idea is to check flow control and divide application programs into general pieces, i.e. dividing it into free cuts of a branch of the code. For each piece, a definite signature has been calculated and errors can be detected by comparing the execution time signature. In many flow control methods, one major problem is setting and adjusting the granularity test.

In [16], portable checkpoints of a compiler have been proposed source by source which can automatically insert instructions from the application program for storing and retrieving from portable check points. Portable checkpoints are capable of having a mode of calculations in an independent form of the machine which is regarded as a potential solution for an error-tolerant software for developing networks of binary machines.

In [18], a systematic method was proposed for introducing data and redundancy code to the available source code which was written in C programming language. Simple rules for changing code was proposed in [18] which is effective for optimizing reliability. However, it is not automatic and it proposes high overhead in terms of memory and execution time.

Benso et al [27] proposed a novel method for computing critical variables in application programs of software. In this method, the analysis of execution time in the behavior of variables was used instead of error injection which is time consuming. For detecting critical variables, the variables which were read more than other variables were used or the variables with identical access rate, i.e. all the variables read or written with the same frequency were proposed. The chief advantages of the model were formalization, high accuracy of the results and also, low calculation time.

Benso et al [20] proposed a robust c/c++ source-by-source compiler for enhancing the reliability of application programs. Strategic detectors are based on two main methods for re-ordering code and duplicate variables. In this research, RECCO tool which has a fully automatic process was used. According to code change, RECCO allows user to exchange between the optimization level of reliability and performance degradation. The introduced changes by the tool are completely obvious for programmers and they cannot influence the main capabilities of the target program. Furthermore, this tool allows users to select the percentage and rate of duplicate variables so that overhead can be maintained under the respective limitations. The experimental results indicate the effectiveness of the method and low overhead in the reliable code in terms of memory occupation and execution time.

III. Proposed Method

In this paper, critical data or data vulnerable to errors were first identified. Then, a limited number of the identified data were duplicated to protect them against error. Figure 1 illustrates a conceptual model of the proposed method.

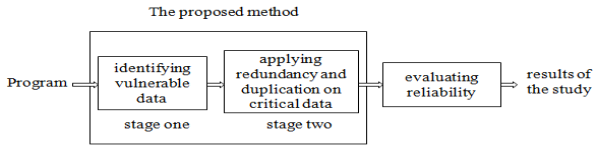

Fig.1. Conceptual Model of the Proposed Method.

The first stage is static. Indeed, by analyzing the static program and the designed model, we tried to identify data vulnerable to errors. In the second stage, for each selected data, the related instructions are duplicated. In fact, by duplicating a limited number of instructions, the program is protected against error. In this way, less performance overhead is injected into the program.

For investigating the proposed method and evaluating its effectiveness on error identification, experimental method was used. Thus, the proposed method was implemented on a number of programs written in the c# programming language. These programs included: auto telling system, elevator system, library robot system, payroll system and artificial intelligence system. Next, by conducting experiments on error injection and the statistical analysis of it, the effectiveness of the proposed method was investigated.

In this paper, a designed model was used for detecting and identifying vulnerable data which was aimed at illustrating the class diagram of the intended program. Using the designed model of the class diagram, all the classes, methods, variables and the relationships between the program classes are specified. The following formula can be used for determining the degree of vulnerability of the classes:

$$V_i = (N_v + N_m + (IM \times IL) + (C_1 \times CR) + (C_2 \times DR) + (C_3 \times AR) + (C_4 \times ASR)) \quad (1)$$

In this formula, V_i indicates the intended class vulnerability, N_v denotes the number of related class variables, N_m refers to the number of related class methods, IM indicates the number of variables and the methods inheritable from the related class, IL denotes the inheritable level from the intended class, CR refers to the number of combined communication and relation to the intended class, DR stands for the number of relations dependent on the intended class, AR indicates the number of relations accumulated in the intended class and also, c_1,c_2,c_3, c_4 which respectively have the coefficients of five, four, three and two indicate the communication or relation coefficient of the intended class.

The proposed method was used to obtain the vulnerability of each intended class and the class which was planned to be worked on was determined in advance.

Table 1 shows a sample of the vulnerability values of the classes related to the Automated Teller Machine.

Table 1. Vulnerability of the ATM's Classes

Classes	Vulnerability
BankCustomer	20
Account	20
ATM	18
ATMCard	10
Transaction	9
CardScanner	6
CashDispenser	5
DisplayScreen	4
CurrentAccount	2
SavingsAccount	2

As the vulnerability of each program class was determined based on the proposed method, the redundancy and duplication on critical data were applied. This operation helped to identify probable changes of the critical data of the program. In the proposed method, redundancy was applied on variables and operators. Figure 2 depicts a way in which redundancy was applied on critical data. Based on the proposed method, about 70% of the program classes including high vulnerability were selected and the redundancy was applied on the critical data of those classes. Using the proposed method, the minimum redundancy was implemented on the critical data of the program.

```
public void test()
{
    int a,a1,b,b1,c,c1,c2;
    a = 101;  <-- Main data
    a1 = 101;  <-- Redundant data
    b = a + 11;  <-- Main data
    b1 = a1 + 11;  <-- Redundant data
    if (a != a1 || b!=b1)  <-- Conditional instruction for the presence of error in data or instruction
    {
        Console.WriteLine("Error");
    }
    else
    {
        c = a + a;
        c1 = b * b;
        c2 = c + c1;
        Console.WriteLine(c2);
    }
}
```

Fig.2. The Way of Applying Redundancy on Vulnerable Data.

After the implementation of redundancy on the vulnerable data of the related classes, the reliability of the program should be guaranteed. Hence, program reliability was evaluated. By evaluating the reliability of the main program, the redundancies of the vulnerable data of the programs were compared with one another in terms of memory consumption and the execution time of the program. The comparison of the program with 70% redundancy and duplication of the critical data and the one with 100% redundancy and duplication indicated that the proposed method had better performance in terms of memory consumption and the execution time. Further explanations are given later in the results section.

IV. Experiments

The present study was based on experiment. For evaluating the proposed method, an extensive set of experiments were carried out. In the conducted experiments, a number of trial programs written in c# program were used. The programs used in this paper included: automatic telling machine, elevator system, library robot system, payroll system and digital intelligence system.

After considering each of the intended programs, the first thing that was done was to draw class diagram for each of them. Having obtained the intended class diagram of the program, all the communications, classes and program data are illustrated. As shown in figure 3, the class diagram of the Automated Teller Machine was an instance of the respective example. As the classes of the Automated Teller Machine were determined and the variables and methods of each class in the intended program were specified, the relations between each class with the other classes can be determined now.

All the measures including the specification of variables and the methods of each class and the relations of classes with one another were taken for obtaining the class diagram of the intended program in the c# programming language. This language was regarded as the intended experimental environment for achieving the results of the study.

By obtaining the class diagram in the c# experimental environment, a table with several classes was considered for gaining the vulnerability of the program classes. It was produced in the Word 2007 software and the vulnerability values of each of the program classes were determined in the table. Table 1 shows a sample of this table.

The codes of each of the considered applications were written based on their class diagram in c# program. As mentioned before, by writing the code of each of the respective programs and obtaining the class diagram for each of them, the vulnerability value of each class can be obtained based on the proposed method. After specifying the vulnerability of the classes, the type of class having critical data is identified and duplicated on the operation data. Critical data was duplicated on 70% of the related class. Then, redundancy was carried out on the critical data of each of the intended programs.

Each of the three programs was evaluated. In evaluating each main program which, in turn, has a 70%-redundancy program and a 100%-redundancy program, the memory values and execution times of each respective program were obtained in the c# program. Then, the Excel 2007 was used for illustrating the evaluation diagram of each program.

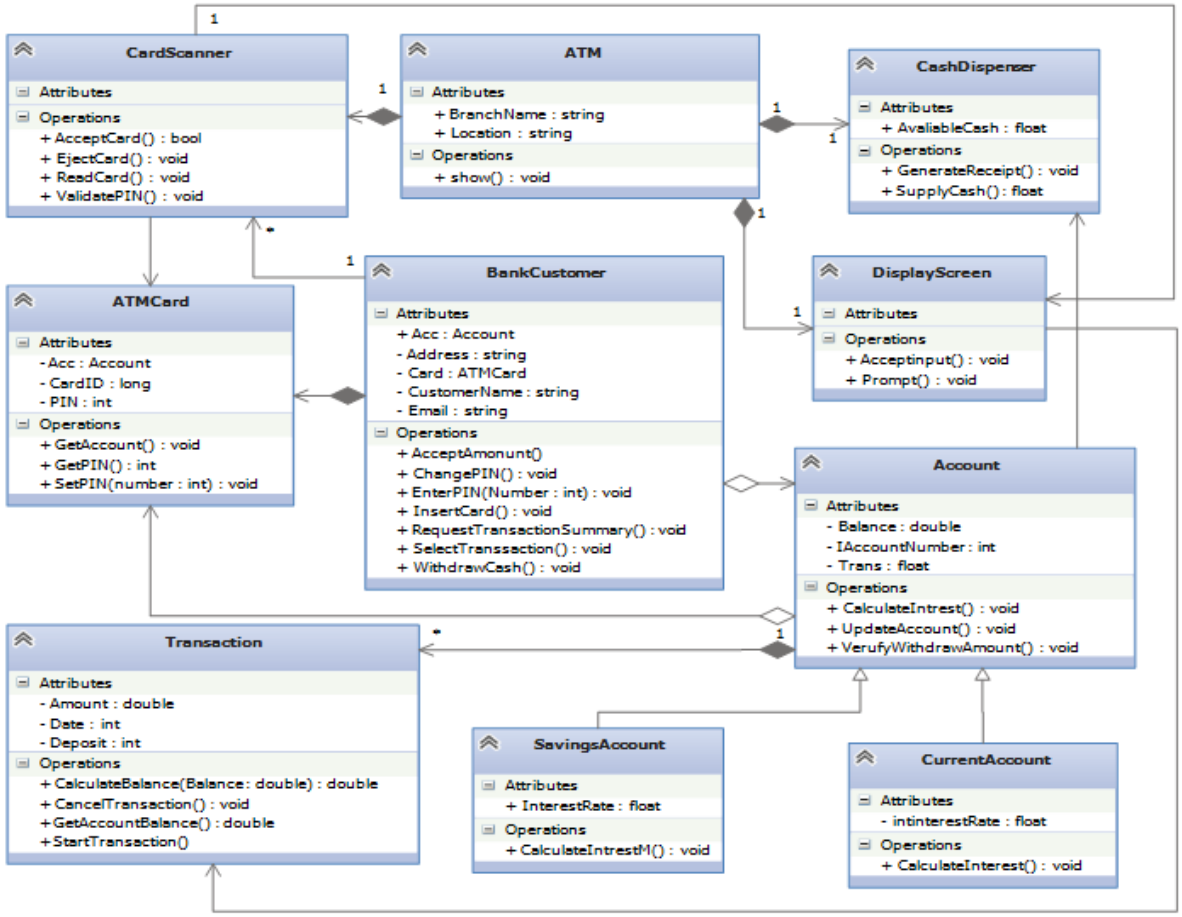

Fig.3. Class Diagram of the Automatic Telling Machine (ATM)

V. Results and Discussion

By obtaining class diagram, the programmer is able to observe the program which will be written in the form of code. More importantly, critical data of the program will be obvious and noticeable via class diagram for the programmer using proposed method. Then, the programmer can consider more reliability for the program data which are more critical and critical and have a greater impact on other classes.

As mentioned earlier, for establishing reliability for the program, the vulnerability of each of the related classes should be gained through the proposed formula. In the proposed formula according to the previous related studies, the relations having more significance among the program classes were taken into consideration. Grading was considered among the assumed relations and a coefficient was specified for each one. This proposed formula helps user to better notice a high-significance class relation. Altering obtaining the vulnerability of each class and applying only 70% of the classes with high vulnerability, it should be proved that there are classes within the 30% of all cases for which it is not necessary to apply redundancy. Thus, the proposed method includes 70% of the classes for redundancy or data duplication. Indeed, redundancy has a remarkable impact on performance overhead which can reduce 30% of the memory consumption and the execution-time of the program.

In the present study, for strengthening the critical data of the program against transient faults, data redundancy was used. In applying redundancy or data duplication on the critical data of the program, in fact, the operation was focused on the variables and operators. Since critical data are so critical and important and, in many cases, they are probably prone to failure and faults, the study zoomed in on those data. Another significant issue which should pointed out here is that, regarding redundancy of critical data, the proposed method considered 70% of redundancy on them in order to protect them.

Since class diagram was used in the proposed method for detecting critical data, it can be maintained that the proposed method significantly enjoys high detection ease and speed. Inasmuch as execution code was used in most related studies for detecting critical data, identifying critical data was very challenging for the programmer and in some cases, it was so difficult to reach an accurate conclusion and proper decision about them. Hence, in general, it should be noted that there is useful information in the design model of software architecture which is not readily and easily extractable and recognizable at the code level. Such information which is used for detecting classes and vulnerable parts of the program is given below:

- Displaying modes of execution time.
- Displaying the type and degree of relations.
- Displaying data aspect, control aspect and the structural aspect.
- Displaying the complexity of different parts of the program.
- Displaying the relations between variables and different models of the program and their degree.
- Displaying variables and different classes of the program without unnecessary details.

The designed model for the proposed method is class diagram. The vulnerability cases of a class include the followings:

- The class from which other class inherit has is assumed to have more critical data.
- The class with more variables and methods.
- The class having more relations with higher relation coefficient.

After obtaining the values of consumption memory and execution time for each of the three modes of the program, i.e. fully duplicated program, 70%-duplicated and the main program, the diagrams of the memory consumption and the execution time of the three modes of the respective program were compared with one another. The followings are the diagrams of the consumption memory and execution time for each of the five programs with three modes. Figures 4 to 11 show the memory overhead (consumption) introduced by proposed method in different programs.

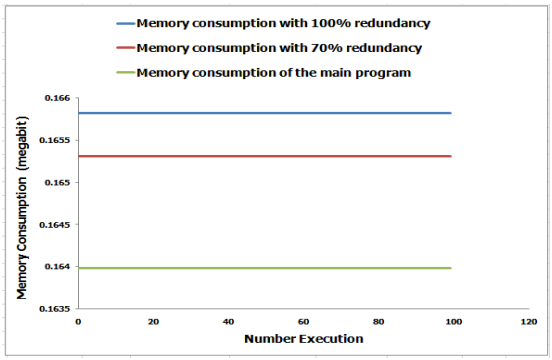

Fig.4. Memory Consumption of the Digital Intelligence System

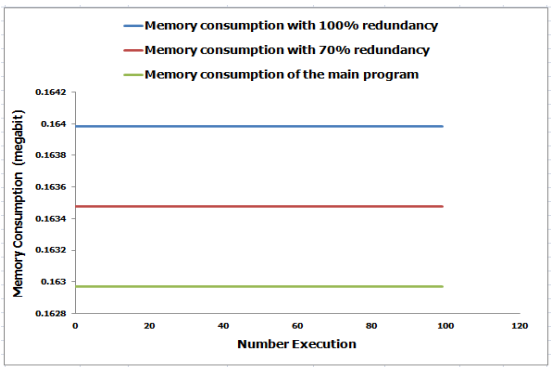

Fig.5. Memory Consumption of the payroll system

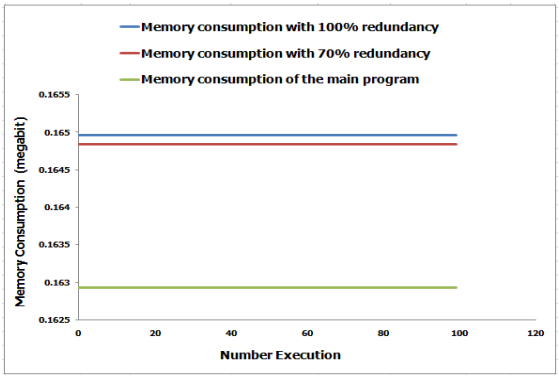

Fig.6. Memory Consumption of the library robot system

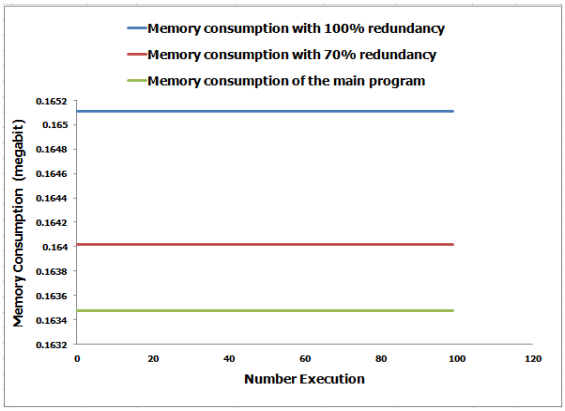

Fig.7. Memory Consumption of the elevator system

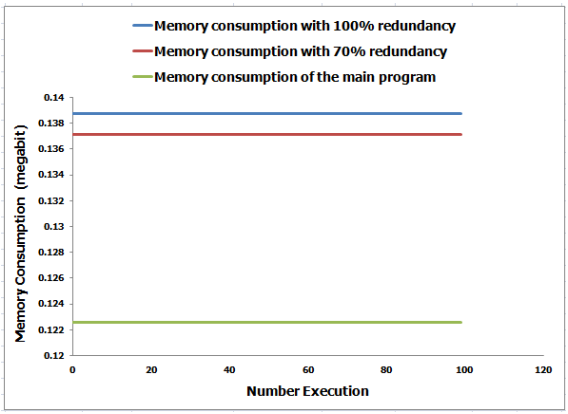

Fig.8. Memory Consumption of the automatic telling machine (mode1)

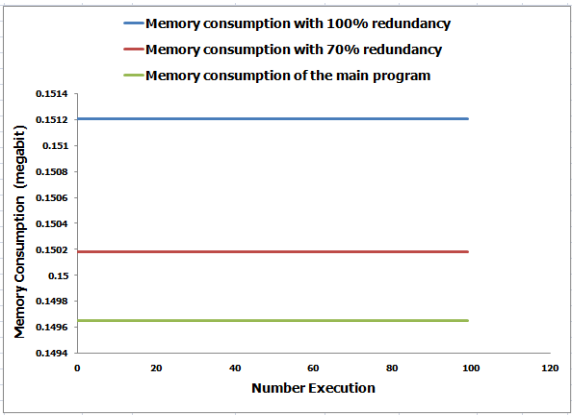

Fig.9. Memory Consumption of the automatic telling machine (mode2)

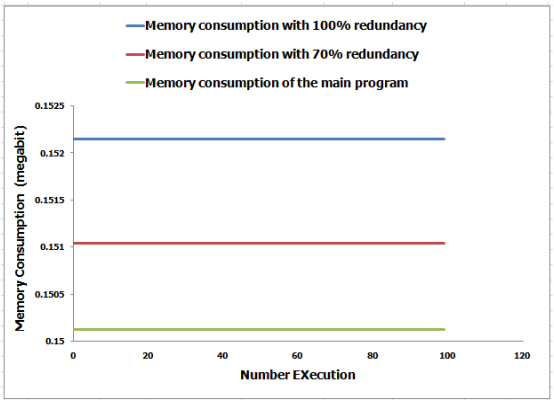

Fig.10. Memory Consumption of the automatic telling machine (mode3)

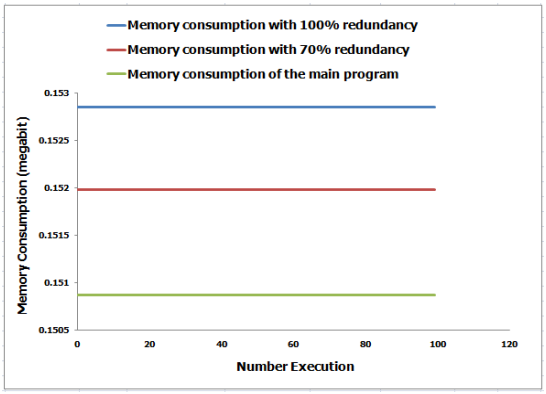

Fig.11.Memory Consumption of the automatic telling machine (mode3)

In the proposed method, 70% of classes (data and instructions) of a program (as the critical classes) are protected against soft-errors; and about 30% of the memory consumption is reduced with regard to the full duplication method. Figures 12 to 19 show the execution time introduced by proposed method in different programs.

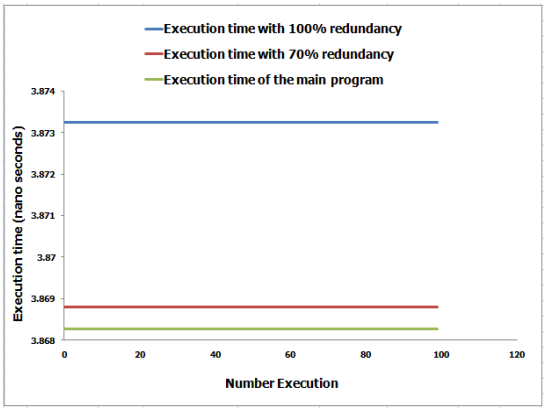

Fig.12. Execution times of the Digital Intelligence System

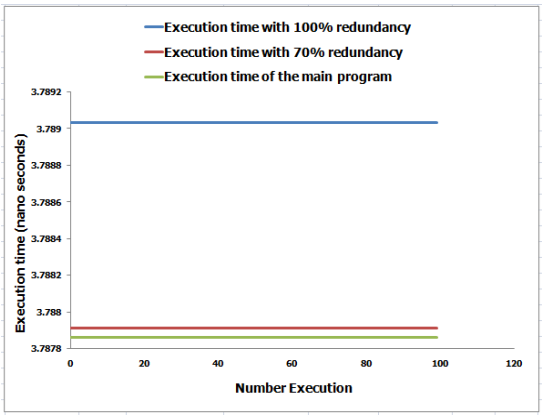

Fig.13. Execution times of Payroll System

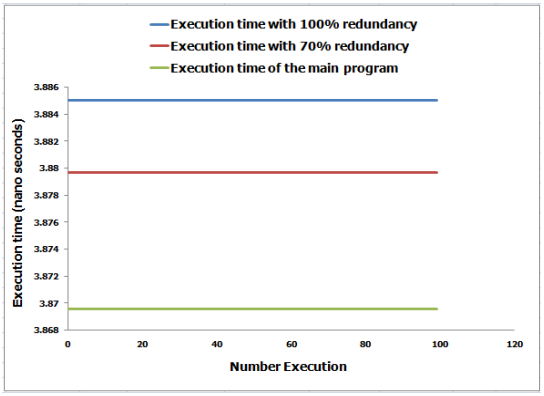

Fig.14. Execution times of the Library Robot System

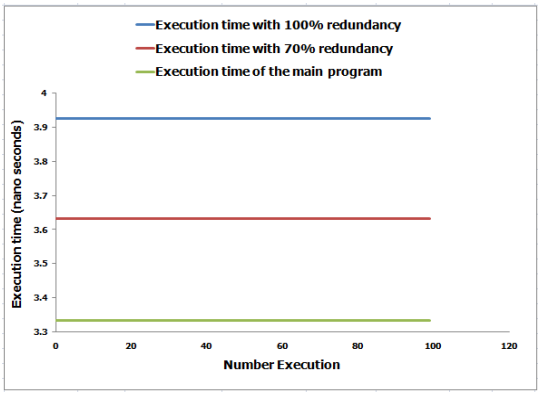

Fig.15. Execution times of the Elevator System

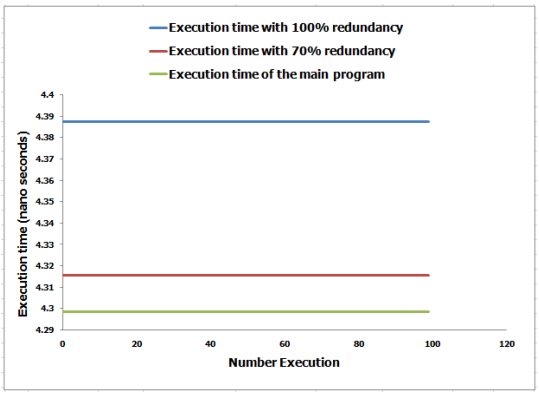

Fig.16. Execution times of the Automatic Telling Machine (mode1)

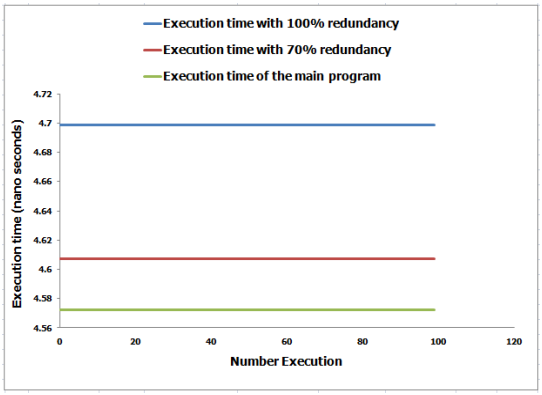

Fig.17. Execution times of the Automatic Telling Machine (mode2)

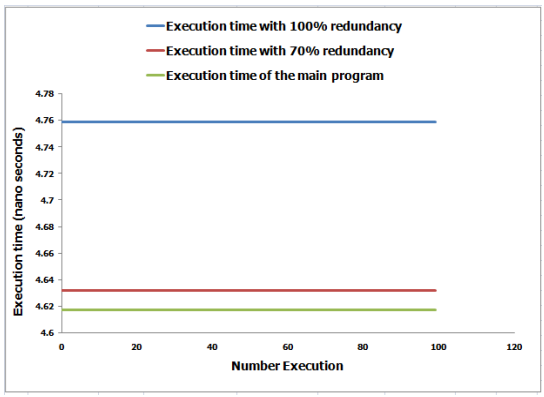

Fig.18. Execution times of the Automatic Telling Machine (mode3)

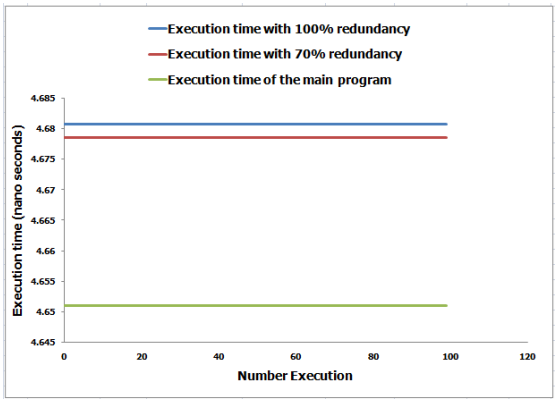

Fig.19. Execution times of the Automatic Telling Machine (mode3)

In the proposed method, the program is protected but 30% of the program execution time is reduced with regard to the full duplication method.

For demonstrating the degree of the efficacy of the proposed method in terms of reliability, each of the respective programs was investigated and analyzed. Hence, at first, in the critical data of all the three modes, i.e. the main program, program with 70% redundancy and the program with 100% redundancy, some errors were injected in the variable values or their operators. Error injection was realized in this way that variable value or the operator sign was changed. The operation of changing critical data is, indeed, similar to errors which have been imposed from outside into the program and lead to failure or fault in the critical data of the program. Also, it results in the failure in the system and program. Then, by executing all the three modes of the intended program, namely the main program, the program with 70% redundancy and the program with 100% redundancy, for which error injection has been applied, the outputs of the programs were compared with one another. The comparison of each of the three modes of the program is given in table 2. It shows that, due to injecting errors, these different modes of the program suffer from certain degree of fault or failure.

Table 2. Evaluation of Program Failure after Injecting Error

Program name	Main program	70% - redundancy program	100% - redundancy program
Auto telling machine (mode 1)	70% to 100%	48% to 57%	25% to 35%
Auto telling machine (mode 2)	85% to 100%	67% to 72%	38% to 42%
Auto telling machine (mode 3)	78% to 100%	52% to 65%	45% to 55%
Auto telling machine (mode 4)	60% to 100%	35% to 45%	15% to 20%
Digital intelligence system	83% to 100%	37% to 53%	17% to 25%
Elevator system	75% to 100%	45% to 55%	12% to 20%
Payroll system	80% to 100%	43% to 57%	15% to 35%
Library robot system	100% (infinite loop)	48% to 52%	25% to 35%

By injecting 10 to 100% errors in the critical data of the intended programs, the results of evaluating program failure were obtained which are given in table 2.

VI. Conclusion

The study reported in this paper was aimed at enhancing program reliability. The method proposed in this paper was intended to secure the critical data of the program such as variables and operators against errors and faults. Class diagram model was used in the method introduced in this study. As discussed earlier, class diagram plays a significant role in obtaining program vulnerability. In the class diagram model, the overall structure without unnecessary details can be illustrated which was efficiently used in line with the purpose of this study. In the designed model of software architecture, there are useful information which cannot be easily extracted and understood at the code level. If we compare class diagram with program code in terms of obtaining program vulnerability, we will find that class diagram which has the merits of observing all the program content, high detection speed and lack of complexity is significantly more efficient and desirable than program code.

The formula proposed in this paper was aimed at obtaining the vulnerability of the class of a program. It was highly beneficial and useful in selecting critical and efficient data and relations. It obviated the need for examining and dealing with unnecessary issues. By obtaining consumption memory and execution time for

the three modes, i.e. 70% redundancy program, 100%-redundancy program and the main program, we found that the main program is more prone to failure and fault due to the lack of redundancy in its critical data. consequently, it has less reliability, consumption memory and execution time. Regarding 100% redundancy program, it was noticed that all the critical data are more reliable against fault and error and resistant to failure. However, it should be pointed out that it has higher consumption memory and execution time. In the proposed method, i.e. 70% redundancy, not only the critical data of the program are protected but also 30% of the consumption memory and the program execution time were reduced. Hence, the performance overhead was consequently reduced. The lack of 30% redundancy in the proposed method was related to the class which was less vulnerable than the other classes.

For evaluating the failure of each of the three modes, errors were injected. The results of the error injection experiments indicated that the main program had less reliability; nevertheless, the 70% redundancy program had 70% reliability and the 100%-redundancy program had 100% reliability. The factors of execution time, consumption memory and program failure analysis via error injection were investigated and compared for the three modes of the program. The results of the comparisons revealed that the program with 70% redundancy based on the proposed method had less execution time and memory consumption and more reliability. Also, the performance was optimized.

In this paper, the experiments were carried out on five different programs by injecting 10 to 100% errors in the three different modes of the program. As mentioned before, execution time, memory consumption and the degree of the failure for each program mode was specified.

As a direction for further research, it is recommended that the critical instructions of a program in addition to the variables and operators should be taken into consideration for enhancing its reliability. Reliability enhancement will be a combination of program codes and software architecture so that the critical data of the program can be accurately covered and be secure and resistant against faults and errors.

REFERENCES

[1] D. J. Lu, "*Watchdog Processor and Structural Integrity Checking*" , IEEE Transaction on Computers, vol. C-31, No. 7, pp. 681-685, July 1982..

[2] J. H. Patel et al., "*Concurrent Error Detection in ALUs by Recomputing with Shifted Operands*", IEEE Transaction on Computers, vol. C-31, No. 7, pp. 589-595, July 1982.

[3] K. H. Huang, J. A. Abraham, *Algorithm-Based Fault Tolerance for Matrix Operations*, IEEE Trans. Computers, vol. 33, pp. 518-528, Dec 1984.

[4] David L.Parnas, A. John van Schouwen, and Shu Po Kwan, *Evaluation of safety-Critical Software Communications of the ACM*, Vol. 33, No. 6, pp. 636 – 648 , June 1990.

[5] K. Wilken, J.P. Shen, "*Continuous Signature Monitoring: Low-Cost Concurrent-Detection of Processor Control errors",* IEEE Transaction on Computer Aided Design, vol. 9, NO. 6, pp. 629-641, June 1990.

[6] H. Madeira, J. G. Silva, "*On-line Signature Leraning and Checking",* Dependable Computing for Critical Applications 2, Springer-Verlag, pp. 395-420, 1992.

[7] T. Goradia, *Dynamic Impact Analysis: A Cost-Effective Technique to Enforce Error-Propagation*, ISSTA 1993.

[8] Jean-Claude Laprie, *Dependability of Computer Systems: Concepts, Limits, Improvements*, pp 3,4, 1995.

[9] Y. M. Hsu et al., "*Time redundancy for error detecting neural networks*", Proc. IEEE Int. Conf. on Wafer Scale Integration, pp. 111-121, Jan. 1995.

[10] Barry W. Johnson, *An Introduction to the Design and Analysis* of *Fault-TolerantSystems*, in Fault-Tolerant Computer System Design, Dhiraj K. Pradhan,Prentice Hall, Inc., pp. 1 – 87, 1996.

[11] A. M. Amendola, A. Benso, F. Corno, L. Impagliazzo, P. Marmo, P. Prinetto, M. Rebaudengo, M. SonzaReorda, *Fault Behavior Observation of a Microprocessor System through a VHDL Simulation-Based Faukt Injection Experiment*, EURO-VHDL'96, Geneva(CH), pp. 536-541, September 1996.

[12] J. G. Silva, J. Carreira, H. Madeira, D. Costa, F. Mreira, *Experimental Assessment of Parallel System*, Proc. FTCS-26, Sendaj(J), pp. 415-424, 1996.

[13] M. ZenhaRela, H. Madeira, J. G. Silva, *Experimental Evalution of the Fail-Silent Behavior in Programs with Consistency Checks*, Proc. FTCS-26, Sendaj(J), pp. 394-403, 1996.

[14] Roger S. Pressman, *Software Engineering: A Practitioner,s Approach*, The McGraw-Hill Companies, Inc., 1997.

[15] V. Strumpen, *Portableand Fault-Tolerant Software System*, IEEE Micro, pp. 22-32, September-October 1998.

[16] V. Strumpen, *Portable and Fault-Tolerant Software Systems*, IEEE Micro, pp. 22-32, September-October 1998.

[17] J. Voas and K. Miller, *The Avalanche Paradigm: An Experimental Software Programming Technique for Improving Fault Tolerance*, Proc. of ECBS, 1999.

[18] M. Rebaudengo, M. Sonza Reoda, M. Torchiano, M. Violante, Soft-error Detection through Software Fault-Tolerance techniques, DFT'99: IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, November 1-3, New Mexico, USA, pp. 210-218, Albuquerque- 1999.

[19] W Torres-Pomales, *Software Fault Tolerance: A tutorial*, pp6, 2000.

[20] A.Benso, S.Chiusano, L.Tagliaferri, *A C/C++ Source-to-Source Compiler for Dependable Applications*, Politecnico di Torino, Dipartimento di Automatica e informatica, pp71-78, 2000.

[21] Lui Sha, *Using Simplicity to Control Complexity*, University of Illinois at Urbana-Champaign, pp 20, 2001.

[22] Laura, L.P.*Software Fault Tolerance Techniques and Implementation*. Boston London: Artech House, pp 1,3,9-12 , 2001.

[23] A. Benso, S. Carlo, G. Natale, L. Tagliaferri, and P. Prinetto, *Validation of a Software Dependability Tool via Fault Injection Experiments*, 7th Intl. On-Line Testing Workshop, 2001.

[24] M. Ernst, J. Cockrell, W. Griswold, and D. Notkin, *Dynamically Discovering Likely Program Invariants to*

Support Program Evolution, IEEE Trans. on Software Engineering, 27(2), 2001.

[25] M. Hiller, A. Jhumka, and N. Suri, *On the Placement of Software Mechanisms for Detection of Data Errors*, Proc. Intl. Conference on Dependable Systems and Networks (DSN), 2002.

[26] A. Benso, S. Di Carlo, G. Di Natale, P. Prinetto, L. Tagliaferri, *Data Criticality Estimation in Software Applications*, Politecnico di Torino, Dipartimento di Automatica e Informatica, Corso DucaDegli Abruzzi 24, I-10129, Torino, Italy, pp 802-810,2003.

[27] A. Benso, S. Di Carlo, G. Di Natale, P. Prinetto, L. Tagliaferri, *Data Criticality Estimation in Software Applications*, Politecnico di Torino, Dipartimento di Automatica e Informatica, Corso Duca Degli Abruzzi 24, I-10129, Torino, Italy, pp 802-810, 2003.

[28] Shubhendu S. Mukherjee, Joel Emer, Steven K. Reinhardt, *The Soft Error Problem: An Architectural Perspective*, pp 1-5, 2005.

[29] S. Narayanan, S. Son, M. Kandemir, and F. Li, *Using Loop Invariants to Fight Soft Errors in Data Caches*, Proc. Asia and South Pacific Design Automation Conference (ASP-DAC'05), 2005.

[30] KarthikPattabiraman, ZbigniewKalbarczyk, and Ravishankar K. Iyer, *Application Based Metrics for Strategic Placement of Detectors*, Center for Reliable and High-Performance Computing, Coordinated Sciences Laboratory, University of Illinois at Urbana-Champaign, pp 1-8, 2005.

[31] Eduardo Valido-Cabrera, *Software reliability methods*, *Technical University of Madrid*, 2006, pp 1-2, 12.

[32] S. Pontarelli, M. Ottavi, A. Salsano, *Error Detection and Correction in Content Addressable Memories*, Rome, ITALY, pp 423, 2010.

[33] Gurpreet Kaur, Kailash Bahl, *Software Reliability, Metrics, Reliability Improvement Using Agile Process*, 2014, pp 143-146.

Authors' Profiles

Saeid A. Keshtgar received master degree in computer engineering from Islamic Azad University of Tabriz. His research interests include Fault Tolerance and reliability of software systems.

Bahman Arasteh received his master degree from IAU of Arak, Iran, in 2006 and the PhD degree in Software Engineering from IAU University, in 2014. Currently, he is an assistant professor in the department of computer engineering at the IAU of Tabriz. His research interests include Dependability and fault tolerance of software systems, Software architecture of dependable systems, Software-based Fault-Injection, Software Testing and validation methods.

How to cite this paper: Saeid A. Keshtgar, Bahman B. Arasteh,"Enhancing Software Reliability against Soft-Error using Minimum Redundancy on Critical Data", International Journal of Computer Network and Information Security(IJCNIS), Vol.9, No.5, pp. 21-30, 2017.DOI: 10.5815/ijcnis.2017.05.03

