Indian Journal of Science and Technology, Vol 9(18), DOI: 10.17485/ijst/2016/v9i18/93779, May 2016
ISSN (Print) : 0974-6846
ISSN (Online) : 0974-5645

Providing an Approach to Locating the Semantic Error of Application using Data Mining Techniques

Abdollah Rahimi[1*], Ahmad Faraahi[2] and Azam Ghaedi[3]

[1]Department of Computer Engineering, Bandar Abbas Branch, Islamic Azad University, Bandar Abbas, Iran; Rahimi.abdollah12@gmail.com
[2]Department of Computer Engineering and Information Technology, Payame Noor University, Iran; Ahmad.faraahi1987@gmail.com
[3]Department of Computer Engineering, Bandar Abbas Branch, Islamic Azad University, Bandar Abbas, Iran; Azam.1999.ghaedi@gmail.com

Abstract

Regardless of the efforts taken to produce a computer program, the program may still have some bugs and defects. In fact, the larger and more complex programs are more likely to contain errors. The purpose of this paper is to present an approach to detect erroneous performance of application using clustering technique. Because the program paased different execution paths based on different inputs, there is impossible to discover all errors in the program before delivery the software. Monitoring all execution paths before delivery of program is very difficult or maybe impossible, so a lot of errors are hidden in the program and is revealed after delivery. Solutions that have been proposed to achieve this goal are trying to compare the information in the implementation of the program to be successful or unsuccessful which called determinant and introduces the points suspended to the error to programmer. But the main problem is that the analysis carried out at the decisive time information regardless of affiliation between predicate, leading to the inability of these methods to detect certain types of errors. To solve these problems, in this paper a new solution based on behavior analysis and runtime of executable paths in the form of taking into account the interactions between determinants are provided. For this purpose, a clustering method was used for classification of graphs based on the similarities and the ultimate determination of areas suspected of error in the erroneous code paths. Assessment of the proposed strategy on the collection of real programs shows the success of the proposed approach more accurate in detecting errors compared to previous.

Keywords: Clustering, Dynamic Analysis, Data Mining, Locating Error, Semantic Error

1. Introduction

Regardless of the efforts taken to produce a computer program, the program may have bugs and defects. In fact, the larger and more complex programs are more likely to contain errors. Always effective elimination of errors for programmers, without inadvertently create new forms, has been a challenge. In addition, developers for debugging must be able to identify exactly where the errors are; known as fault localization and then find a way to solve them, which is known as fault fixing[1].

Localization software fault is one of the most expensive activities in debugging the program. Fault localization can be broken into two broad categories:

First category is using a method to detect the malicious code may contain bugs.

Second category is testing and identification code and makes a decision about whether this part of the code is really a fault or not, that is the responsibility of the programmer.

All solutions of localization error are focused on the first part to prioritize the malicious code based on the potential value that contains fault[2].

* Author for correspondence

2. Define the Problem

Fault localization methods generally are divided into two categories: static and dynamic. Static methods try to plan based on dependency graph (as a reference model) to detect fault location within the program[3,4]. In contrast, the dynamic methods are trying to compare the performances of successful and unsuccessful program to identify the approximate location of fault[5]. Collecting the data needed to model the pathways program is a major issue in the discussion of debugging software.

In practice, save all the administrative information produced by the program is not possible and only a part of the program can be saved. To resolve this problem, only parts of the program that can detect application performance and behavior are identified and treatment is based on data obtained from the analysis of these points during the performances.

These points are called predicate which can be true or false value. Instrumentation is a procedure during which the information by adding code to the original program code for determining execution time is obtained[6].

In the process of anomaly detection and fault locating dynamic analysis methods are of great importance, especially techniques based on predicate evaluation[7]. In this methods using Instrumentation decisive values at runtime (successful, failed) program collected for analysis. Save the information necessary to assess all routes runtime executable program, but in practice this is impossible. Solution of this is the same as determining the values of certain parts of the program are collected and modelling is done based on application behavior.

The purpose is detection of erroneous software performance after software fault is occurring. To do this with the help of a model's behavior is assessed. Select areas of the program that their values should be gathered in a way that the evaluation of performance in these areas, anomalies in the implementation of the program is detected. Faults in the program usually appear in the program branch and return values of functions, therefore, these points can be considered as Predicate[3,8].

In this study, we watn to answer the following questions: What do the data mining techniques applied in locating faults programs can have meaning? Is the proposed method for localization faults of the required application in the detection and localization multiple errors?

For detecting faults in the software using clustering techniques, little work has been done[9]. Despite efforts to find hidden errors program, but there are still many bugs and errors that can be problematic or even some of them are associated with adverse consequences and high costs. One of the most costly stages of software development that much time is allocated to the implementation of the procedure is to locate and debug errors.

This article seeks to prove or disprove the hypothesis that the clustering performance of multiple errors in the application program can be used to detect and localization. Clustering can be used to distinguish unsuccesssful behaviors, and then the program failed to localize the errors.

The purpose of this paper is to provide a method for detecting erroneous application performance using clustering technique. Because the program is based on different inputs, different administrative paths are passed, before delivery software to users discover all the errors is not possible. Check all running paths before delivery program is very difficult or maybe impossible, so a lot of errors in the program remain hidden and is revealed after delivery.

There are many methods for detection of latent semantic errors in software. According to the mentioned problems, the approach presented in this paper is in different parts of the software to detect errors and provide new solutions.

2.1 Description of the Problem and Solutions

The aim is to find the place or places suspected fault code. 50% to 80% of the software development process activities reduce the number of error. One of the actions required to reduce the number of fault, debugging is the most time consuming. The hardest part of debugging is to localization faults[10].

Various methods are provided to debug the software and discover the different types of errors[11]. Among the presented methods, statistical methods have had a lot of success. These methods are run-time information on the performance of successful and unsuccessful, in certain parts of the program that the program branch is normally referred to as Predicate.

Then a statistical analysis on this data, identify determinants suspected fault. The main problem is that these methods are not able to detect complex faults on several predicate. The main reason is that these methods have the same effect on the result of the application will not be considered.

2.2 Statistical Methods of Fault Localization

These methods use the relationship between probability and statistics and have high accuracy to debug software. Performances are based on successful and unsuccessful programs, information gathering programs, and by using techniques based on probability and statistics, derived and ranked based on the probability of erroneous fault Predicates of being suspected[12].

In statistical methods should be to deliver the software to the user, the program will be implemented in large numbers. The program must be implemented before delivery for different inputs the data collection program in all possible execution paths. Of course performance data collection program is completely impossible in practice and even if the action may be required to spend a lot of price.

2.3 Architecture Proposed Approach

The purpose of the proposed solution is to identify errors in the program with a behavioral model. In order to assess the status of implementation of the program, the status of implementation of the program as a point in an n-dimensional space is displayed. In other words, any program can be considered as a point in space, so if a program run n times, such as every time we run a point in space, the space program is an n-dimensional space. Then using clustering techniques, we classify these parts.

The proposed solution is shown in Figure 1. In instrumentaion stage, some codes are added to the program to save the predicate value in the running time.

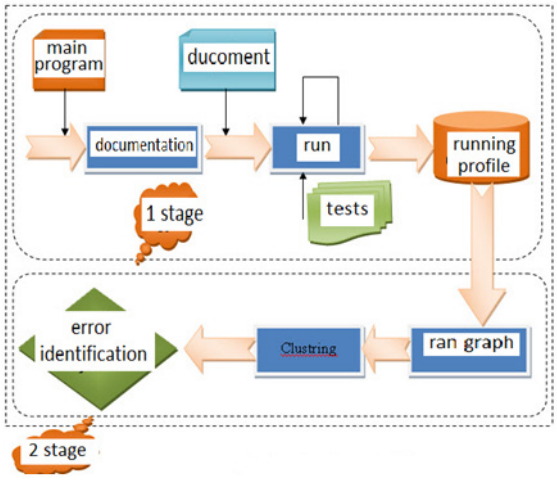

Figure 1. Architecture of proposed method.

Predicates can be true or false. For each performance, a vector is derived from predicates. Of the on these vectors are generated from the implementation of performance graphs. Finally, by doing clustering and the definition of a suitable standard for the primary purpose of acquiring clusters that we recognize Predicates is erroneous.

2.4 The First Stage of the Proposed Approach

The first stage is the proposed approach consists of two main parts:

(1) Instrumentation and

(2) Apply

In stage Instrumentation code is added to the program to run-time information can be stored. There is no possibility of collecting all this information, even if the data can be collected, it is difficult to check. Parts of program should be limited and only data points collected. Conditional sentences and return values are good points for Instrumentation. In effect, the program's Instrumentation for different cases was examined.

Most important action takes place in the implementation phase is predicates that be translated into a form that can be displayed. How to store predicate information is important because we assume that the behavior of predicate points reflects the program behavior. In fact, we want to recognize abnormal behavior of the program using information obtained by predicate behavior.

3. Form Sequence Predicates

When the error occurs in a different point in program may be affected by this fault. According to studies, the best places to detect errors in the program are conditional statements and functions return values.

In a conditional statement predicate influence on each other predicates in flow direction control on production. Predicates behavior can be comapred in success or failure in performance can be compared and predicate whose behavior is different in successful and unsuccessful performances are announced as predicates suspected of fault[13]. This method has been used in several studies. But given the dependence predicates and excluding the impact of other predicates, criterion for assessing the implementation of a program can not be successful or unsuccessful.

To better understand the contents of the code shown in the form of a program that insertion sort algorithm to calculate the number of reps with a certain number of entries in the array. There are eight predicate labels P_1, P_2, P_3, P_4, P_5, P_6, P_7 and P8 in this piece of code. For sorting arrays [] = (10, 13, 8, 1, 12) and if number 2 is read from the input number predicates the order of their true value will be as follows (figure 2). Thus different performances during the program, several sequels will be made to predicates.

P1 P1 P1 P1 P1 P2 P3 P2 P3 P2 P2 P3 P3 P4 P4 P4 P4 P4 P5 P5 P5 P5 P5 P7

Figure 2. Contents of the code.

According to this sequences derived from the performance of the programs, some graphs are made that represent the behavior of the program of the beginning to the end of its run. Then, how to create graphs based on the sequence of predicate is described.

3.1 The Second Stage of the Proposed Approach

The second stage consists of two components: the creation of executive graphs and clustering. The most important activity done at this stage is clustering. To reduce the overhead of the data collection program in the study, the structure of the graph has been used. The remainder of this section first and then examines how to create graphs executive and clustering of successful and unsuccessful performance of the program will be discussed.

3.2 Create Executive Graphs

After the program documented run-time information encoded in a sequence, a file was saved. Since the execution paths of the ring and returning calls can be very long and voluminous, therefore these administration paths can be displayed using graphs executive.

For this purpose, the predicate can be selected as a node or vertex of the graph between two nodes p and q edge will be executed if p then q also weight of the

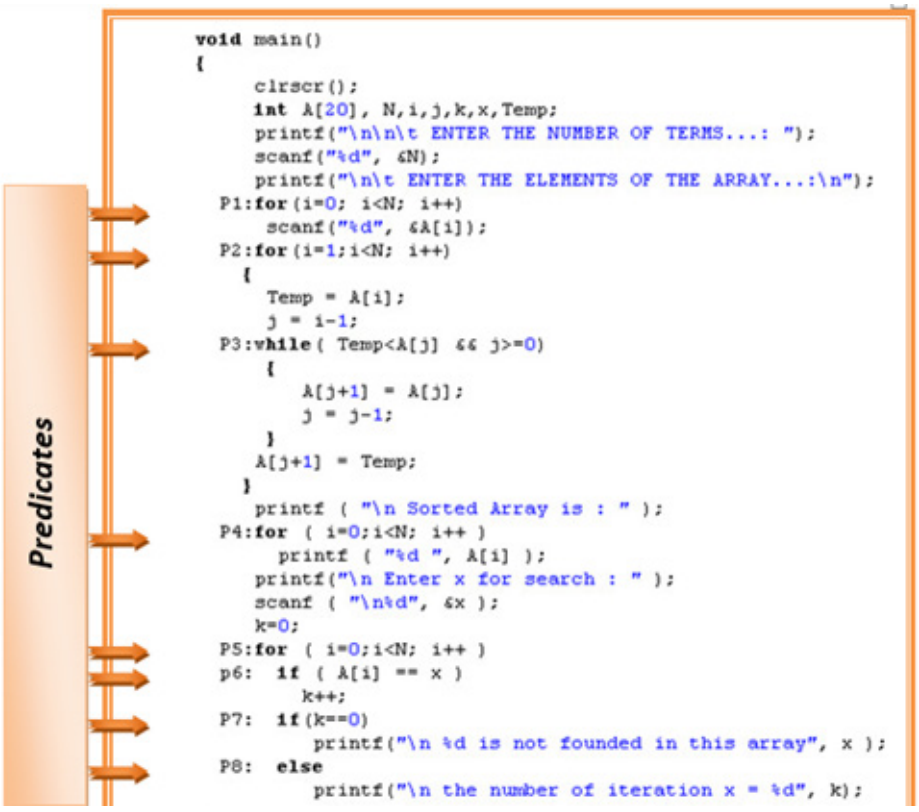

Figure 3. Specifying the predicates.

edge between p and q is the number of times after that q p is true. For understanding, graphs pertaining to the program are displayed in the form of arrays array [] = (10, 13, 8, 1, 12).

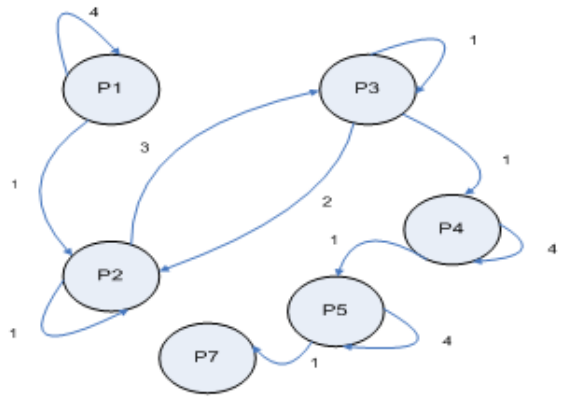

Figure 4. 12 performance graph of the sample program.

3.3 Clustering

After gathering data, runtime, aims to recognize successful and unsuccessful performances with the highest similarity. For this purpose in this section, a clustering method based on a similar function is provided. Selected implement clustering to perform the following steps:

Select a distance measure

Choose a clustering algorithm

The data divided into a number of clusters in such a way that the data set to display correctly.

3.4 Data Collection Stage

Data collection stage consists of three stages: instrumentaion, implementation and run-time and data collection. First, parts of the program in behavior analysis programs are effective to be selected as predicate. In stage Instrumentation code is added to the program to store predicates information.

4. Instrumentation

Studies show that most errors when there is a change in the way programs are implemented. So in this thesis, conditional statements and functions as a predicate return values have been used. Predicate as a conditional statementcan have two values either true or false[14]. To accurately evaluate our performance during the run, the predicate condition is divided into two predicate. Each predicate is a path for the executable program.

5. Implementation and Data Collection

After Instrumentation and add code to the main program P, instrumentation the program 'P is obtained. As explained in the description of the proposed approach for a sample application, with performances Instrumentation in every program, a sequence T, the identification number that is assumed for any predicate obtained and then stored in a file. predicate may be any number of zero or more, for each program, the number of times that each assessment is predicate the correct amount, identification number must be stored in the file.

6. Clustering Stage

Clustering stage consists of two phases: production executive graphs and detecting errors. In the first section the implementation of predicate identity that every time programs are evaluated instrumentation with the correct amount, were stored in a file. Now, information in the form of sequence T executive paths has been collected.

7. Assessment

In this paper, we evaluate the proposed solution, the Siemens test programs have been used in C code. With changes, the proposed approach can be used to debug software with any programming language. Siemens is the source SIR. Estimates show that the programmer to detect fault, compared with methods developed in this area need to check the less volume of code.

8. Test on Siemens

Siemens test includes 132 erroneous versions of 7 different programs, each version there is an average of one fault in these programs has been created manually. In Table 1 statistical profile 7 has been shown.

Table 1. Statistical Profile Siemens test suite

Name of program	erroneous copies	Number of lines	All cased of test	Program explanation
Print_tokens	7	539	4130	Lexical analyzer
Print_tokens2	10	489	4115	Lexical analyzer
Replace	32	507	5542	Pattern replacement
Schedule1	9	397	2650	Priority scheduler
Schedule2	10	299	2710	Priority scheduler
tcas	41	174	1608	Altitude separation
tot_info	23	398	1052	Information measure

8.1 First test on Siemens

The purpose of this test is to determine the ability of the proposed approach in operating software to detect erroneous performances. Figure shows the effectiveness of the proposed approach. As shown in the following figure with increasing distance executive graphs successful and unsuccessful programs, the numbers of errors detected are reduced. Whatever performance graphs are similar, the distance will be less.

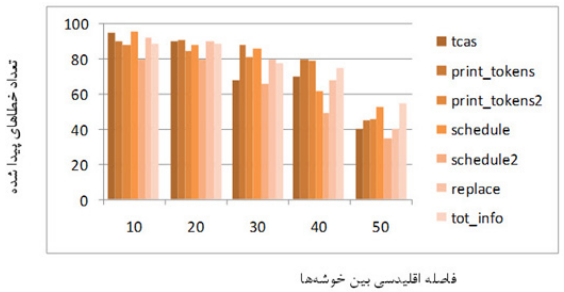

Figure 5. Percentage of fault found by the clustering in Siemens test.

8.2 Second test on Siemens

The purpose of this test is to demonstrate the accuracy of the proposed approach in identifying predicates erroneous. In this test, seven Siemens program is used. Copies of each application is selected, then instrumentation programs and collect information about the performance of each version of erroneous applications and finally apply the proposed approach on the information collected in the tables below have been predicates of erroneous detection.

Columns of the table are as follows: Name of program, number of copies of the program that are instrumentated, the number of successful performances, the number of unsuccessful performances, predicate that there is really fault, predicate which errors are suspected. Predicate that fault in the column are really dark line is placed, to this means that the error has been created by removing part of the code that these errors are not instrumentation identification number. In column predicate the impact of such fault on other predicates suspected of fault is shown.

Table 2. Results obtained of the print tokens 1

Name of program	Number of copy	Number of successful running	Number of unsuccessful running	Predicate with real error	Predicates suspected to error
1 Print tokens	1 V	4124	6	15 P	15 P-14 P
1 Print tokens	2 V	4082	48	15 P	15 P-14 P

Table 3. Results obtained of the print tokens 2

Name of program	Number of copy	Number of successful running	Number of unsuccessful running	Predicate with real error	Predicates suspected to error
Print tokens 2	1 V	3875	240	-	24 P- 13 P 56 P- 13 P
Print tokens 2	3 V	4082	33	-	25 P- 13 P 24 P- 13 P

Table 4. Results obtained of the Schedule 1

Name of program	Number of copy	Number of successful running	Number of unsuccessful running	Predicate with real error	Predicates suspected to error
Schedule1	1 V	2643	7	23P	P35 – P23 P21 – P4

Table 5. Results obtained of the Schedule 2

Name of program	Number of copy	Number of successful running	Number of unsuccessful running	Predicate with real error	Predicates suspected to error
Schedule 2	2 V	2679	31	-	P85 – P75
Schedule 2	5 V	5381	39	-	P7 – P19
Schedule 2	8 V	2642	68	64 P	P64 – P21 P64 – P44

9. Conclusion

Much of the software development process activities are related to software debugging. Traditional debugging methods usually involve two steps:

(1) Show unexpected behavior

(2) Run test cases to find the cause of the error

Such an approach is challenging debugging, tedious and error-prone, because the space is very large and fully accessible. Fault localization step towards automated debugging: the amount of code that is not related to the fault will be filtered and only the remaining code should be examined.

Automated methods are not only more accurate than traditional methods, but also save a considerable amount of their time. So we can conclude that debugging automated methods are preferred over the traditional methods. When coding fault in the program will try to place some important work to do:

(1) Identify the instructions that are involved in the fault.

(2) Statements that may contain fault are suspected.

(3) Identify the fault

Due to the irreparable damage caused by errors in the software may be time to recognize the important latent semantic faults in software. Many researches have been done in the field of software debugging, but none of them can guarantee to find all semantic faults in program before its delivery to the user. So, debug software can be regarded as an important research topic.

10. References

1. Haji Baba M, Parsa S. Localization hidden faults Application using cross entropy and N models. Journal - Promoting Soft Computing. 2013; 3:44–59.
2. Zeller M. Why Program Fail: A Guide to Systematic Debugging, 1st ed. San Francisco: Morgan Kaufmann, 2006.
3. Hangal S, Lam M. Tracking down software bugs using automatic anomaly detection. Proceedings of the 24th International Conference on Software Engineering. 2002. p. 291–301.
4. Montazeri-Gh M, Mahmoodi-k M. Development a new power management strategy for power split hybrid electric vehicles. Transportation Research Part D: Transport and Environment. 2015 Jun; 37:79–96.
5. Chebaro O. Program slicing enhances a verification technique combining static and dynamic analysis. Proceedings of the 27th Annual ACM Symposium on Applied Computing. ACM, 2012.
6. Li J, Weiss D. Efficient program instrumentation. U.S. Patent No. 2013; 8:616–23.
7. Zhiqiang Z, Khoo S. Iterative Statistical Bug Isolation via Hierarchical Instrumentation. 2014; 3:129–36.
8. Thiyagarajan VS. Platform Method for High Data Delivery in Large Datasets. Indian Journal of Science and Technology. 2015; 8:343–51.
9. Podgurski A, Leon D, Francis P, Masri W, Sun MM, Wang B. Automated support for classifying software failure reports. Proceedings of the 25th International Conference on Software Engineering. 2003.
10. Diehl S. Software visualization: visualizing the structure, behaviour, and evolution of software. Springer Science and Business Media. 2007; 8:25–32.
11. Brun Y, Ernst M. Finding Latent Code Errors via Machi Learning over Program Executions. Proceedings of 26th Int'l Conf software Eng (ICSE '04). 2004; 480–90.
12. Montazeri-Gh M, Mahmoodi-k M. An optimal energy management development for various configuration of plug-in and hybrid electric vehicle. Journal of Central South University. 2015; 22(5):1737–47.
13. Parsa S, Ebrahimi K, Narey N. Provide a mechanism for locating errors on graph semantic software. Sixteenth Annual International Conference of Computer Society of Iran. 2010.
14. Agrawal R. design and development of data classification methodology for uncertain dat. Indian Journal of Science and Technology. 2016; 9:143–53.