IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661, p- ISSN: 2278-8727Volume 16, Issue 3, Ver. V (May-Jun. 2014), PP 35-38
www.iosrjournals.org

A Novel Approach for Code-Base Fault Localization Technique

Suvajit Mandal, Arup Abhinna Acharya

Abstract: *The relationship between testing and debugging is intimate one. Finding the location of a fault is a central task of debugging. Typically, a developer employs an interactive process for fault localization. To improve the quality we have to remove as many defect as possible in it without introducing new bugs at the same time. However localizing a fault is a complex and time consuming process. There is many different procedure to resolve the problem to locate the exact fault in a program. We can use the model-base and code-base scenario using Slicing and SAFL. Finding a fault in a program is a complete process which involves understanding the program's purpose, structure, semantics and the relevant characteristics of a failure producing tests.*

Index Terms: *Testing, Debugging, Slicing, SAFL.*

I. Introduction

The relationship between testing and debugging is intimate one. Though testing requires understanding not only on the program implementation. To improve the quality we have to remove as many defect as possible in it without introducing new bugs at the same time. However localizing a fault is a complex and time consuming process. Once, the execution of program fails, various debugging techniques and tools are employed to locate the bugs. Typical debugging process is to locate the error code first, then to replace those faulty statements with correct once. The statistical fault localization techniques may use their corresponding execution statistics to pinpoint suspicious program entities, understanding the trade off due to the use of test case prioritization strategies on the effectiveness of fault localization techniques.

Finding a fault in a program is a complete process which involves understanding the program's purpose, structure, semantics and the relevant characteristics of a failure producing tests. Finding the location of a fault is a central task of a debugging technique. Typically developer employs an interactive process for fault localization. Fault localization is a critical task for tester which assurance process, in order to reduce testing cost and improve availability ,reliability and performance of a application or overall system. . Fault localization can reduce the delivered faults by the program itself and locate the fault, so that developer can easily understand that what correction process they should have to take. Software fault localization is not much different from fault localization in any other domain. Instead, we apply the same principle to diagnose software as in any other system . To improve the quality of a program , we have to remove as many defects as possible in it without introducing new bugs at the same time. However, localizing a fault is a complex and time consuming process. Mark Weiser introduced program slicing of error variable to execute irrelevant statement thus to reduce the searching domain. Much information is available to help us localize a fault after software testing, such as testing requirements and their associated test case, test result etc. while most fault localization technique have not taken into consideration.

II. Basic Concept

There are some key concept and depending upon that we can localize the fault in any system. These are the basic requirements we need to examine before the localization. These are as follows:

A. Anomalous Behaviour: Faulty system show many anomalies apart from the actual failing output. They tend to show behaviour much different from the standard correct behaviour.

B. Experimentation: Experimenting with a system can help gain information about the fault.

C. C. Dependence Analysis: System have cause- effect chains. These cause effect chains can be traversed to find regions containing fault. .

D. Knowledge Reuse: Previous knowledge about a system can be useful to locate the fault. This knowledge could be in various forms, like earlier experiment with resolving faults or learning system specific properties gain new information from the system. This process repeats till we localize the fault.

III. Methods For Fault Localization

There exist few different method which is applicable on different field of software state. The main methods are as follows:

1. For Model-Based-SAFL:-

The basic idea of SAFL is to use the execution information to deal with the distribution of test cases over statements in the program. In the debugging process, each testing executed may include different statements, so the statement contributes differently to the result.

2. For Code-Based-Slicing:-

It is well-known that as the Size of program increases, it becomes impractical to maintain them as monolithic structure. Indeed, splitting programs in smaller pieces allows to construct, understand and maintain large programs much more easily. Program slicing, is a program manipulation technique that extracts from programs statements which are relevant to a particular computation.

IV. Related Works

W. Eric Wong et al.[1] proposed an approach on fault localization using Dstar method. Here he stated fault localization technique named Dstar is proposed based on a modification of the Kulczynski coefficient. Effectiveness of Dstar is evaluated across 21 programs, and compared to 16 different fault localization techniques. It is demonstrated that Dstar is better. Yu-Min Chung et al.[2] proposed an approach that is testing-based fault localization approach (TBFL), is a kind of debugging technique to locate faults and generally utilizes the information which could be acquired from testing, such as information of coverage and execution results. Such a method locates the faults by the analysis of a large number of execution traces. Jirong Sun et al.[3] proposed an approach on fault localization using execution slicing. In this paper an execution dice is the set of basic blocks or decisions in one execution slice which do not appear in the other execution slice. A concept very close to our use of execution dicing for fault localization.

V. Proposed Work

we have prescribed our approach in a model which contains no of steps. First step is code, means we have to have a faulty code and the test cases are applied on this code .So there after we will get some execution path respectively for each test case. Among those execution path some of them are right and others are wrong. We will select some test cases as our requirements from those results and will apply our formula to choose the most suspicious domain. The model is given below-

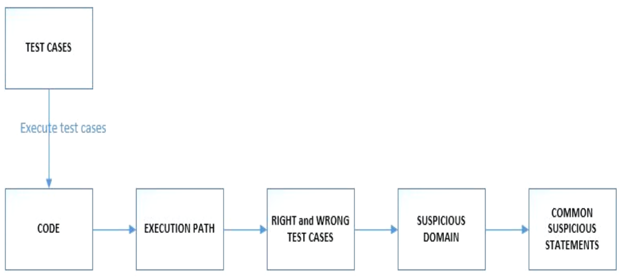


Figure 1: Proposed framework

Proposed steps are as follows-

step1. Code: This is the code for which we have to find the fault.

step2. Test case: A test case is a set of conditions or variables under which a tester will determine whether a system under test satisfies requirements or works correctly. For a given code P, we need a test case or test suite that is relevant to the code. But for our code the test cases or test suite are already given.

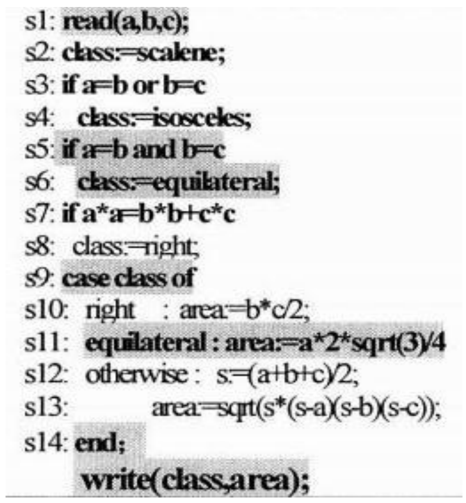
step3. Execution path: After executing the test suite we can find the execution path. For our code that has already given.

step4. In this step our major duty to select out the right and wrong test cases. The test cases giving the exact result in accordance with the code ,is called the right test cases and which are unable to show the exact result is called the wrong test cases.

step5. Finally we have to calculate the most suspicious domain in the code by using P 0 = Ewt.

A. Case Study

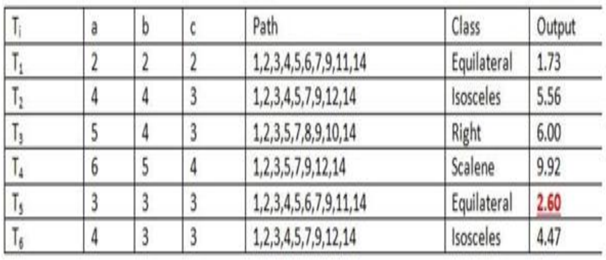
Sample Program



```
s1: read(a,b,c);
s2: class:=scalene;
s3: if a=b or b=c
s4:   class:=isosceles;
s5: if a=b and b=c
s6:   class:=equilateral;
s7: if a*a=b*b+c*c
s8:   class:=right;
s9: case class of
s10:  right    : area:=b*c/2;
s11:  equilateral : area:=a*2*sqrt(3)/4
s12:  otherwise :  s:=(a+b+c)/2;
s13:            area:=sqrt(s*(s-a)(s-b)(s-c));
s14: end;
     write(class,area);
```

The above program calculate the area of Scalene, Isosceles, Right angle and Equilateral triangle. From sample code or program .we can see that there are three inputs a,b and c. For each of them we have to give different inputs to get the output as class and computed area and also the execution path. T1, T2 , T3, T4 , T5, T6 are the test cases. Now we can construct the below table

Table 1: Output and execution path by each test case



T_i	a	b	c	Path	Class	Output
T_1	2	2	2	1,2,3,4,5,6,7,9,11,14	Equilateral	1.73
T_2	4	4	3	1,2,3,4,5,7,9,12,14	Isosceles	5.56
T_3	5	4	3	1,2,3,5,7,8,9,10,14	Right	6.00
T_4	6	5	4	1,2,3,5,7,9,12,14	Scalene	9.92
T_5	3	3	3	1,2,3,4,5,6,7,9,11,14	Equilateral	2.60
T_6	4	3	3	1,2,3,4,5,7,9,12,14	Isosceles	4.47

From the above table we can examine that the underlined output is wrong one for its given input. So T5 is a wrong test case. Here we taking three right test case from the table which give the exact result with the formula. Now, we can represent the three right test cases T1, T2, T3 as- execution path E1, E2 , E3 respectively. In the other side the wrong test case is denoted by Ewt .From the output table it is clear that, the T5 results a wrong output. Therefore Ewt = T5. The execution path for T5 is 1,2,3,4,5,6,7,9,11,14. Now here we are discussing the basic concept of dynamic slicing a little. There are mainly two concept. They are-

(a) If a statement is not executed under a test case, it can not affect the program output for that test case.

(b) Even if a statement is executed under a test case, it does not necessarily affect the particular output.

From the above we can observe that, for T5 execution, the statement no 8,10,12 and 13 did not reach at all. Therefore, these are not affecting the output for T5. Another important observation is that, statement no 2,3,4,7 are executed under T5,but they can not affect the variable area .So we can discard these statement line from T5. So, now T5 becomes 1,5,6,9,11,14. Now, we are going to propose some equation using basic and simple mathematical intersection and union properties. These are as follows-

1. $E123 = E1 \cap E2 \cap E3$.
2. $E12 = E1 \cap E2$
3. $E1+2+3 = E1 \cup E2 \cup E3$.
4. $E1+2 = E1 \cup E2$

From the standard definition of intersection we can understand that E123 contains the least no of codes. Now, we are introducing the most suspicious domain in the code that is P 0. Here P 0 is equivalent to the first wrong test case T5. So P 0 = 1,5,6,9,11,14. Our formula to find fault is as follows:

p5 =p0-E1+2+3

p4 =p0-E1+2

p3 =p0-E1
p2 =p0-E12
p1 =p0-E123

Now, from our previous discussion we know that intersection of the three execution path contain least no of code. So the probability is less that this part contain bugs also rather than the part of union of the three right test case execution path. Because union will contain more lines of code. So we can conclude this as-
E123 $\leq$ E12 $\leq$ E1 $\leq$ E1+2 $\leq$ E1+2+3
After calculating all the values of P, finally we have to find the value of P 1 ∩P2 ∩P 3 ∩P 4 ∩P5.

B. Results
We know know Ewt = 1,5,6,9,11,14.
There are three right test cases and their three execution paths are-E1 , E2, E3. Now
E1 = 1,2,3,4,5,7,9,12,14. E2 = 1,2,3,5,7,8,9,10,14. E3 = 1,2,3,5,7,9,12,14. So by intersecting E1 and E2 ,we get E12 = 1,2,3,4,5,7,9,12,14. E123= 1,2,3,4,5,7,9,14. E1+2+3 = 1,2,3,4,5,7,8,9,10,12,14. E1+2 = 1,2,3,4,5,7,8,9,10,12,14. P 0 = Ewt.
Thereafter, P 5 = P 0 − E1+2+3 = 1,5,6,9,11,14 − 1,2,3,4,5,7,8,9,10,12,14 = 6,11 P 4 = P 0 − E1+2 = 1,5,6,9,11,14 − 1,2,3,4,5,7,8,9,10,12,14 = 6,11
P 3 = P0 − E1 = 1,5,6,9,11,14 − 1,2,3,4,5,7,9,12,14 = 6,11 P2 = P0 − E12 = 1,5,6,9,11,14 − 1,2,3,5,7,9,14 = 6,11 P 1 = P0 − E123 = 1,5,6,9,11,14 − 1,2,3,5,7,9,14 = 6,11
Here one thing to look out that we are calculating the values of P from the P5 to P 1 gradually. Because E1+2+3 contains more no of code among the others. So the values of P 1 ∩P 2 ∩P 3 ∩P 4 ∩P 5 =6,11. we can easily understand that the fault lies in between line 6 and 11. More precisely the faulty statement is line no 11,because there should be a*a instead of a*2.

VI. Conclusion

Fault Localization is a difficult task. It requires resources and time, so we should try to partially or fully locate faults . Currently fault localization techniques is based on slicing where time and complexity matters , so we have given a framework which will locate faults in a efficient way. Most of the previous work in the field of fault localization is done by using dynamic slicing which contain a large no of codes in it.So our objective is to proposed a method which will more easily identify the fault in a code.

References

[1]. W. Eric Wong, Vidroha Debroy, Yihao Li, and Ruizhi Gao. Software Fault Localization Using DStar (D). In IEEE Sixth International Conference on Software Security and Reliability (2012), vol. 2.
[2]. Yu-Min Chung,Chin-Yu-Huangg. A Study of Modified Testing-Based Fault Localization Method, vol. 2. 2008.
[3].]Jirong Sun, Zshu Li, J. N. F. Y. Software Fault Localization Based On Testing Requirement and Program Slice. In International Conference On Network- ing,Architechture and Storage, IEEE (2007), vol. 1.
[4]. Nimit Singhania. Survey Of Software Fault Localization. In IBM India Research Laboratory, New Delhi,110070,India (2012).
[5].] Dan Hao, Lu Zhang, H. M. J. S. Towards Interactive Fault Localization Using Test Information. In XIII ASIA PACIFIC SOFTWARE ENGINEERING CONFER- ENCE,IEEE (2006), vol. 2.

Suvajit Mandal, M.Tech research scholar school of computer Engineering ,KIIT University,Bhubaneswar
India.

Arup Abhinna Acharya, Assistant Professor at KIIT University,Bhubaneswar,India.