ISSN 2278-3091
Volume 9, No.4, July – August 2020
International Journal of Advanced Trends in Computer Science and Engineering
Available Online at http://www.warse.org/IJATCSE/static/pdf/file/ijatcse365942020.pdf
https://doi.org/10.30534/ijatcse/2020/365942020

Root Cause Detection and Test Case Generation for Student Academics

T.Mamatha[1], B. Rama Subba Reddy[2], C Shoba Bindu[3]
[1]Research Scholar, Dept. of CSE, JNTUA University, Anantapuramu, AP, INDIA, mamathat7@gmail.com
[2]Professor & HOD, Dept. of CSE, SV College of Engineering, Tirupati, INDIA, rsreddyphd@gmail.com
[3]Professor , Dept. of CSE , JNTUA University, Anantapuramu, AP, INDIA, shobabindhu@gmail.com

ABSTRACT

Cause-effect graph concept is used to shows the relation between the effect (outcome) and cause (condition/input). We identified and define the effect and fill the effect box and draw the spine, than constructed binart tree based code events, iterate through the tree to get the result, if the results are on the false side it will backtrack towards to the root to find the actual cause and will print. The work scan the code line by line and will show the line number where actual the problem is. We took one sample test code(Fibonacci series) to demonstrate the concept. Concentrated on the bugs such as library function inclusion, parenthessis matching and package inclusion. Once the root cause is found, these data can be used for test case generation, student academic details are used in the easiest way. We categorized the student's strength levels on the basis of their studying capabilities, regularity in class and participation activities in the Indoor or outdoor level. Depending on this categorization we analyzed the effect, then Graph is constructed, it then converted into a decision table, and lastly we conclude with test case generation. So by considering the effects, the department heads can also take prior mitigations. This study useful for testing people who involved in application related to educational institutions can have the scope to pick up the test cases directly. This paper also helps educational institutions to reason out the actual cause for their effects.

Key words: Cause Effect Graph, Students academics, Decision Tables, Test Case.

1. INTRODUCTION

A student graduate education gives a competitive edge when it comes to searching jobs and earning a living. A higher education also provides students with latest knowledge in areas that interest them most. So In college, students academics place a vital role for college reputation point of you also, in this regard college as well as department will take necessary actions and prepare formats to maintain their academic records high but due to some reasons, deficiencies may exists. To figure out the root cause for those deficiencies this paper will definitely useful.

In order to demonstrate the above we used cause effect graph, which helps to generate test cases. These test cases can be useful for tester to pick up directly for those who involved in projects related to educational background.

2. RELATED WORK

Our previous paper [1] used cause effect graph, which place a vital role in fining the root cause for automatic correction of an error, another work [2] is for college placement process, for this Cause effect graph and decision table helps college students and placement people the procedure of placement and status. Again our work further extend to find the Root Cause[3] with the help of Cause effect Graph and resolved the fixes with the help of fix schemas. The above papers help me to write other paper titled Ranking of fixes for choosing the best fix and the current paper helps the testing and Department Heads of educational institutions.

Research on recommender systems for software bug fixing is part of a big research agenda on software bugs. Currently, research is being done to find new bugs either statically or dynamically. There is still huge research to support the process of handling software bugs. Example 1) how to automatically prioritize bug reports 2) How to assign bugs automatically to competent developers, this stage we reached with advent of testing tools in market. 3) What are the code commits that are related to a given bug report? Then, there is a work on automated debugging which consist of new approaches and tools to help localizing buggy pieces of code ex:fault localization, Finding the root cause, the causal dependencies e.g. with code dependency graphs. Other works implemented within Auto Fix-E [5], give basis for my research work and that works have introduced to leverage specifications inside programs to automatically fix program source code, Pachika Tool, which generate finite state behavioural models from failed and passed test cases. Then the tool generates fixes which changes the failing runs patterns to passing runs. Sometimes tool may delete or insert the existing transaction. For Spreadsheet applications

Abraham and Erwing[7] proposed an idea where for a user given expected value, if it gives wrong value than for an error prone formula it suggest to change the value in the cell to get expected value.

3. CAUSE EFFECT GRAPH

It is one of testing technique that aids in selecting test cases that logically relate Causes (inputs) to Effects (outputs) to produce test cases. A "Cause" queue for an input condition that gets about an internal change in the system. An "Effect" represents an output condition, a system transformation or a state resulting from a combination of causes.

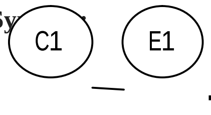
The reasons for using Cause effect graph is:

- To discover the present problem so that correct decision can be taken very fast.
- To narrate the connections of the system with the factors affecting a particular process or effect.
- To recognize the probable root causes, the cause for a exact effect, problem, or outcome.

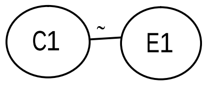
3.1 Symbols and Constraints

The graph technique restates the given requirements specification in terms of logical relationship between the causes and effects conditions. Since it is logical, it is obvious to go for Boolean symbols like AND, OR and NOT.

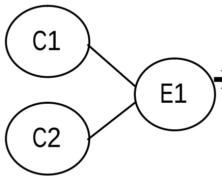
Sy



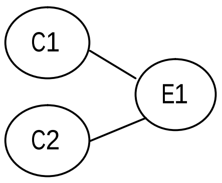
➔identity



➔Not

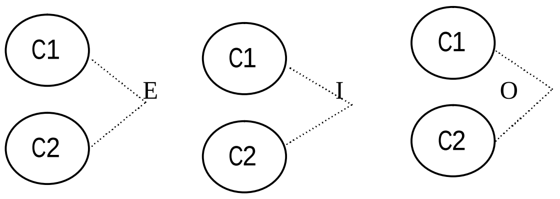


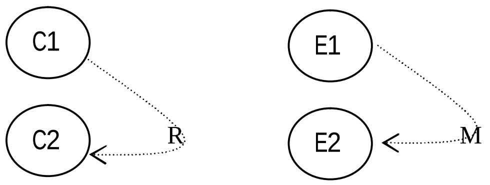
➔AND



➔OR

Contraints:





- The **E(Exclusive)** constraint express that both inputs(Causes) cant be true simultaneously(At most one)

- The **I (Inclusive)** constraint express that at least one of the inputs(Causes) C1,C2 must always true and C1 and C2 cant be false at the same time (At least one)

- The **R (Require)** constraint express that cause C2 require to happen because C1, other words it is impossible to cause C1 to be true and cause C2 to be false

- The **O constraint** express that Exactly one of the cause C1 and C2 can be true.

- The **M (Mask)** constraint express that the effect E1 is true E2 forced to be false

3.2 Algorithm for RootCause Detection

1. Identify and define the causes and effects
2. Construct a binary tree based on events
3. Iterate through the tree to obtain the result
4. Identify the main causes contributing to the effect being studied
5. If the result is on the False side, backtrack towards the root
6. Print from leaf node to the root node

For every event occurred, we store the event in a node of the binary tree. Later, depending upon the truth value, we store the next possible child events on either side of the tree. Once, the generation of the tree is completed, we iterate through the tree from the root node to the solution present in the leaf node. If the result is on the false side of the tree, we backtrack till the root and print it accordingly. Concentrated on the bugs like library function inclusion, parenthessis matching and package inclusion. The below visualizes the work flow

Results → Cause n → Cause (n-1) …... Root cause

3.3 Algorithm for Test Case Generation

1. Recognize and Identify the causes
2. Identify the effects
3. Establish the graph of relation between cause and effects

4. Complete the graph by adding the constraints at appropriate places
5. Convert the graph into a decision table
6. Generate test cases from the decision table

4. PROBLEM STATEMENT

The causes may be thought of as the input to the program, and the effects may be thought of as the output. Usually the graph shows the nodes representing the causes on the left side and the nodes representing the effects on the right side. There may be intermediate nodes in between that combine inputs using logical operators such as AND and OR.

Causes:

C1 : Students Regular in class and good in all subjects(% >70)
C2: Students attention towards lecture is good
C3: Good in co-extra curriculum activities
C4: Moderate in studies and Regular to class
C5: Poor in subjects and not regular in class
C6: Poor in subjects but regular in class
C7: Health issues and other illegal activities

Effects:

E1: Students top with first or Distinction
E2: Second Class
E3: Students Failure
E4: College Topper
E5: All Rounder

Cause Effect Graph:

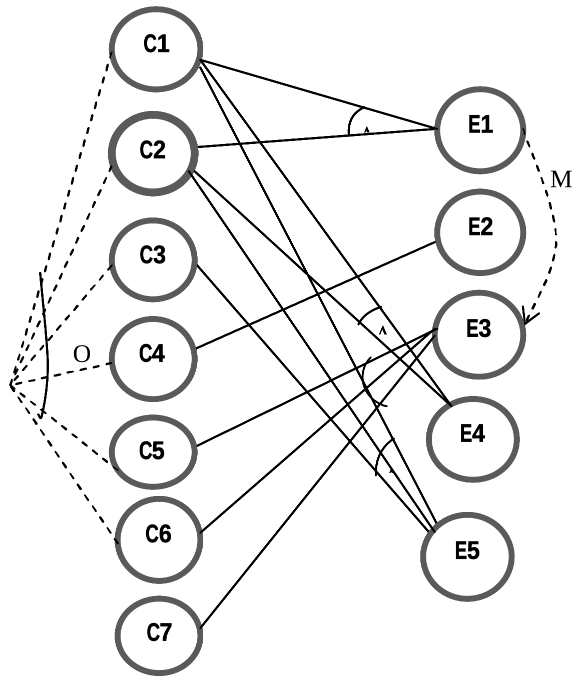


Figure 1: Cause Effect Graph

Decision Table

Table 1: Decision Table

Cause	1	2	3	4	5
C1	T	-	-	T	T
C2	T	-	-	T	T
C3	-	-	-	-	T
C4	-	T	-	-	-
C5	-	-	T	-	-
C6	-	-	T	-	-
C7	-	-	T	-	-
Effect					
E1	T	-	-	-	-
E2	-	T	-	-	-
E3	-	-	T	-	-
E4	-	-	-	T	-
E5	-	-	-	-	T

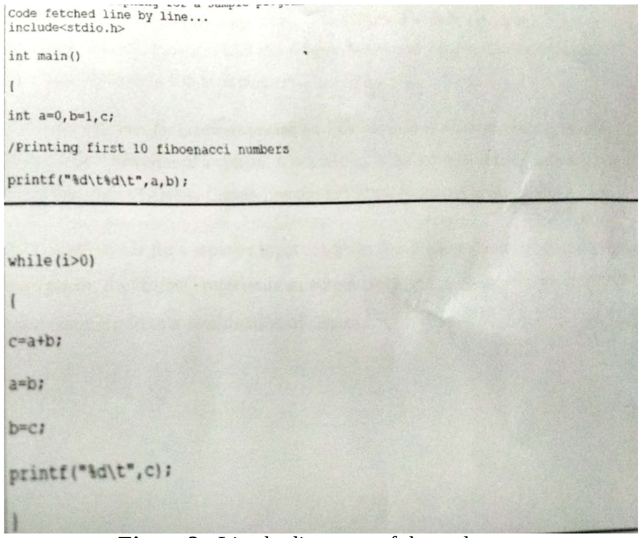
Test Cases Generation

Table 2: Test Cases

Test Caese ID	CAUSES (Test Data)	EFFECTS(Expected Results)
TC01	Students Regular in class and good in all subjects(% >70)	Students top with first or Distinction
TC02	Students Regular in class and good in all subjects(% >70) and Students attention towards lecture is good	College Topper
TC03	1)Students Regular in class and good in all subjects(% >70) and Students attention towards lecture is good 2)Good in co-extra curriculam activities	All Rounder

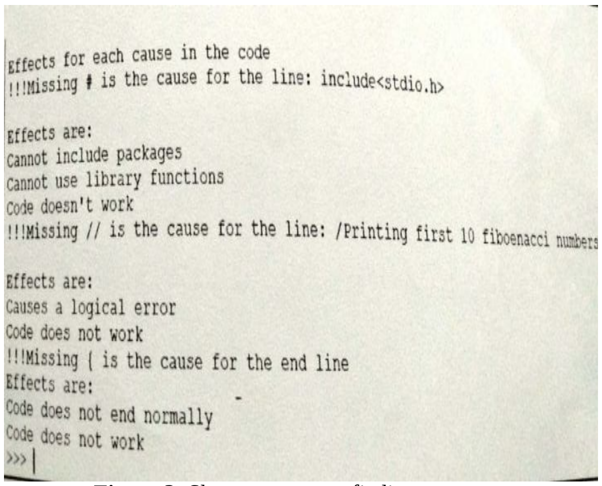
TC04	Moderate in studies and Regular to class	Second Class
TC05	Poor in subjects but regular or not regular in class	Students Failure
TC06	Health issues and other illegal activities	Students Failure

OutPut



```
Code fetched line by line...
include<stdio.h>

int main()

{

int a=0,b=1,c;

/Printing first 10 fiboenacci numbers

printf("%d\t%d\t",a,b);

while(i>0)

{

c=a+b;

a=b;

b=c;

printf("%d\t",c);

}
```

Figure 2 : Line by line scan of the code



```
Effects for each cause in the code
!!!Missing # is the cause for the line: include<stdio.h>

Effects are:
Cannot include packages
Cannot use library functions
Code doesn't work
!!!Missing // is the cause for the line: /Printing first 10 fiboenacci numbers

Effects are:
Causes a logical error
Code does not work
!!!Missing { is the cause for the end line
Effects are:
Code does not end normally
Code does not work
>>> |
```

Figure 3: Shows root cause finding

5. CONCLUSION

Cause-Effect Graph is one of the testing technique which starts with set of requirements and determines the minimum possible test cases for high test coverage which reduces test execution time and ultimately cost. Demonstrated with sample code where its find the root cause of the problem. Figure 2 shows the step by step scan of the code and figure 3 show the line numbers of the root cause. Table 2 shows the test case generation. The study helps software tester who dealing with applications related to educational domain. So the tester can pick up the test cases easily and also give justification for the root causes. It also helpful to the students, teachers and software engineers/ Testing related domain people to understand cause effect graph basic concept.

REFERENCES

1. Ms.T.Mamatha, A.Balaram, Dr.D S R Murthy Automated Detection of Root Cause and Fixing of Programs, International Journal of Electrical and Computing Engineering Proceedings, Vol.1, Issue.4, pp. 20-24, June 2015
2.DhanammaJagli, T.Mamatha, T,Swetha Mahalingam,Namrata Ojha, International Journal of Software Engineering & Applications (IJSEA), Vol.3, No.6, pp. 77-85, November 2012 https://doi.org/10.5121/ijsea.2012.3406
3. Software Testing Foundations:A Study guide for the certified tester exam. Text book by Andreas Spillner, Tilo Linz, Hans Schaefer, 4th edition, Rocky Nook, 2006,pp.266 pages.
4. Mrs. T.Mamatha, Dr. M Chandra Mohan, A BalaRam paper titled Ranking of FIXes for Choosing the Best Fix, , IEEE 7th International Advance Computing Conference, July-2017, pp. 15-20
5.Y. Wei, Y. Pei, C. A. Furia, L. S. Silva, S. Buchholz, B. Meyer, and A. Zeller "Automated fixing of programs with contracts," in ISSTA, 2010. https://doi.org/10.1145/1831708.1831716
6. A. Bachmann and A. Bernstein. Correlations in software engineering datasets. In IEEE Computer Society. MSR'10,Cape Town, South Africa, , pp. 62-71, May 2010.
7. M. Erwig and R. Abraham, proposed "Goal- directed debugging of spreadsheets", IEEE Symposium on Visual Languages and Human-Centri Computing, pp. 37–44, 2005.