A Tool for Generation of Automatic Control Flow Graph in Unit Testing of Python Programs

Akhilesh Kumar Srivastava, Rijwan Khan, Sneha Jain

Abstract: Software Testing (ST) is procedure of discovery out whether the application meet specified requirement or not. It is most important to deliver a reliable and error free software to customer. For the same software should be tested on site. To do this task testing is performed on the software before delivering it. A lot of efforts and money is involved in the testing process. Testing can be done by manual as well as automatically. In automatic testing process unit testing has a role. For the unit testing path testing should be performed. Path testing is a process to check all def-use of variables. So it is more important to generate a correct Control Flow Graph (CFG) for a particular program. In this paper authors develop a tool in Python to generate a CFG automatically for a python program. This tool also finds all the paths from starting to end. This tool will help researchers in unit testing of a particular program.

Index Terms*: *Control Flow Graph (CFG), Software Testing (ST), Unit Testing (UT), Automatic CFG Generator Tool, Path Testing.

I. INTRODUCTION

Manual testing was first performed in the software industries and it became the sole process known to investigate the Software or Programs for bugs. Software testing process requires by enlarge 50% cost of the entire software development process because of the kind of complexity involved in it and its labor-intensive and time consuming nature. Manual Testing process can be automated that reduces the Testing time significantly. There are two broader categorization of Software testing process: dynamic testing and static testing. Static testing progresses by taking the Specification document, Design Document and source code under test as the base. Code written for software gets verified statement by statement without actual execution of the SUT. Hence the process of static testing methodology involves Thus static testing methods are scrutiny, desk checking and evaluation of code. On the other hand dynamic testing involves observation of output after execution of input test data for SUT. The overall quality of Testing process and its significance is directly impacted by the Test case set utilized during the Testing process.

Revised Manuscript Received on June 19, 2019.

Akhilesh Kumar Srivastava, Computer Science & Engineering Department, ABES Engineering College, Ghaziabad, India.

Rijwan Khan, Computer Science & Engineering Department, ABES Institute of Technology, Ghaziabad, India.

Sneha Jain, Computer Science & Engineering Department, ABES Institute of Technology, Ghaziabad, India.

Exposition to ST body of knowledge might facilitate the Software Developers in producing bug free and program to customer's satisfaction and can also impacts positively in the skills of a Programmer. An example will make this clearer, fundamentally all the programs designed have faults [1, 2], this is principle the teachers teach everyone in early lectures of Software Testing. This facilitates the Programmers to become more cautious while writing their codes for a Software. The motivation of the Formal Testing process is to the programmers to write code with diligence. Consider the example of boundary value analysis in wherein the criterion for functional testing mandates the designing of test cases for border inputs. Software developers knowing the test cases for boundary values will make them more cautious while writing the codes. [10, 11, 13, 14].

Although the impact of testing data on programmers seems straight forward but on the contrary there is no empirical proof to justify the theory. Within the literature, we are able to realize substantial work on improving ST coaching in CS discipline. For example, Patterson et al. suggested the mixing up of tools for testing with the coding atmospheres [3]; Jones has proposed the mixing up of testing with entry level CSE courses over testing laboratories and variety of courseware [4]; and Elbaum et al. suggested a e-learning tutorial for engaging the students in learning packages for software testing [5]. Investigations into this subject are necessary. In a result of recent information depicts that computing educational curricula tend to emphasize coding practices without introducing requirement of Testing to be a specific engineering discipline [6, 7, 8]. In fact, as reported by Astigarra et al. [6], the overall CSE curricula tend to position an important stress on style and implementation, instead of on ST like quality assurance subjects. On the opposite hand, even when ST courses are there in the present in curricula, It is quite unknown the extent to which the techniques should be taught in order to be adopted by the industry. (e.g., mutation [9] and data-flow testing [24] appear to be rarely placed for practicing). Lot of studies are present which shows that Software testing education will cause the reliability in Code produced by the programmer units. Especially, it will inspire the designing and up gradations of the courses. In the current paper, authors present two investigations connected with ST learning and software package trustworthiness: The one involves the trainee students and other focuses Educators [20]. The part of student suggested is the experiments with huge data set that addresses and assesses the effect of Software testing education on Reliable Code

outcome, in comparison with supplementary knowledge sharing styles. On the other hand Educators' study focused the survey to adjudge the impact of ST data on educators who take up the fundamental Programming Classes. The focus was to verify (1) if the ST education will play a vital role in programming skills of the learners towards writing the bug free and reliable Programs and (2) if the CS educators/trainers involve a practice of writing the test data in the style of coding they are taking up for teaching. Subjects enforced two completely variant functions prior to and post learning the basic ST paradigm along with the methods (Black Box testing/functional, White Box testing/ structural, Mutation Testing). This way the code standards were compared for before and after effects. The author's goal was to know how ST data might impact on the students'/trainees' coding skills in producing a number of reliable implementations (i.e., testing code's quality itself was not measured, and technique was applied specifically too was unclear). Author do not suggest the verification of extent to which the technique was utilized. To verify the correctness and dependency of the functions utilized, the developed code was executed with test sets developed methodically prior to and later the training was executed.

II. SOFTWARE TESTING AND AUTOMATIC TEST CASES

Software testing is considered utmost essential for S/W development industries as it is bound to deliver the quality software to the client. It also aims at reflecting the extent of user friendliness. A fraction of 50% is spent on the Software testing process in terms of resources for Software Development [15, 16, 17, 18, 19], the process is very lengthy in terms of time as well. Software testing process performs the following important activity apart from doing bug's identification

(i) Enhancement of Software in terms of quality
(ii) Software's validation and verification
(iii) Estimation of Software's reliability

III. PROPOSED SYSTEM

A. Control Flow Graphs

This is a Digraph of which the nodes signify the basic blocks and every edge signify control flow among the basic blocks. Basic blocks refers to sequential instruction sets in which there is only one entry (the instruction appears at very first) and single exit (the instruction that appears last).

B. Core Functioning of Tool

The tool developed takes program (written in python) as input from the user and generates CFG for it. Further, Complexity and all the different paths of the CFG is generated.

Algorithm to Generate CFG:-

Input: - Program
Output: - List containing nodes and pair of edges

1. Extract keywords (print, elif, for, if, else, while) from the program and append them into a list in the following manner:
   a. count = 0
   b. if the extracted keyword is not in nested form, then simply append it into the list along with its count number.
   E.g.:- if3
   count=count+1
   c. if the extracted keyword is not in nested form, then add a unique identity in it along with the count number to identify it easily.
   e.g.:- ifsp3
   For the keywords – elif and if, also add a unique identity for representing the statements they contain and add it into the list.
   e.g.:- if3, ifexp3
   count=count+1

2. At the beginning of the list insert 'start' and at the end of the list insert 'end'.

3. Create nodes for all the items in the list.

4. Create a new list (Suppose list 2).

5. Create an edge between node 1 and node 2 and add this pair in list 2.

6. i=1 , st=0,en=0,stfor=0,enfor=0

7. Create a new list (Suppose list 3).

8. While (i! =len (list))
9. {
10. if ListM[i] contains if , then append in list 3 – ListM[i],ListM[i+1],ListM[i+2] and Create an Edge between nodes – ListM[i] and ListM[i+1].

Also, append the edge pair into ListM 2.

if ListM[i+2] does not contain sp (unique identity for nested keywords) ,then create an edge between node ListM[i] and ListM[i+2] and append this pair in list 2.

```
if ListM[i+2] contains sp , then :
  For j in range i+2 to len(ListM)
        {
                if ListM[j] does not contain sp :
      Break
  }
Edge (ListM[i],ListM[j])
Add in ListM 2 – ((ListM[i],ListM[j]))
Add in ListM 3 – (ListM[j])

if ListM[i+2] does not contain elif and else ,then :
  Add in ListM 3 – (ListM[i+1])
  Edge-
(ListM(i+1),ListM(i+
2))
```

Add in ListM 2 –
((ListM(i+1),ListM(i+2))

if ListM[i+2] constains else and ListM[i+3] does not contains sp , then :
Edge - (ListM(i+3),ListM(i+2)) and (ListM(i+1),ListM(i+3))

Add in ListM 2 –
((ListM(i+3),ListM(i+2)) and ((ListM(i+1),ListM(i+3))

Add in ListM 3 – (ListM(i+1),ListM(i+2) and ListM(i+3))

i=i+1

11. else if ListM[i] contains elif , then
Add in ListM 3 – ListM[i], ListM[i], ListM[i+1],ListM[i+2]

Edge – (ListM[i],ListM[i+1]) and (ListM[i],ListM[i+2])
Add in ListM 2 – ((ListM[i],ListM[i+1])) and ((ListM[i],ListM[i+1]))

i=i+1

12. else if ListM[i] contains elifsp , then
Add in ListM 3 –ListM[i], ListM[i], ListM[i+1],ListM[i+2]

Edge – (ListM[i],ListM[i+1]) and (ListM[i],ListM[i+2])

Add in ListM 2 – ((ListM[i],ListM[i+1])) and ((ListM[i],ListM[i+1]))
if ListM[i+1] contains elifspexp , then :
if(i+1>=st and i+1<=en) , then
Edge – (ListM[i+1],f)

Add in ListM 2 – ((ListM[i+1],f)

Add in ListM 3 – ListM[i+1]
i=i+1

13. else if ListM[i] contains else , then ,
if ListM[i+1] contains sp , then ,
Edge - (ListM[i],ListM[i+1]))

Add in ListM 2 - ((ListM[i],ListM[i+1]))

Add in ListM 3 - (ListM[i])

14. else if ListM[i] contains forsp , then ,
Edge - (ListM[i],ListM[i]))

Add in ListM 2 - ((ListM[i],ListM[i]))

if ListM[i-1] contains for , then ,
Edge-((ListM[i],ListM[i-1])
Add in ListM 2 - ((ListM[i],ListM[i-1]))
else if ListM[i+1] contains sp , then ,
Edge - (ListM[i],ListM[i+1])
else if (i>=st and i<=en):
Edge - (ListM[i],f)
Add in ListM 2 -((ListM[i],f))
else:
Edge - (ListM[i],ListM[i+1])
Add in ListM 2 -
((ListM[i],ListM[i+1]))

15. else if ListM[i] contains for, then ,
f1=ListM[i]
if ListM[i+1] does not contains sp , then ,
Edge - (ListM[i],ListM[i])
Add in ListM 2 - ((ListM[i],ListM[i]))
stfor=i
for j in range - i+1 to len(ListM)
{
if ListM[j] does not contain sp, then ,
enfor=j
break
}

Edge - (ListM[i],ListM[j])
Add in ListM 2 - ((ListM[i],ListM[j]))
if(ListM[i+1]!=ListM[j]) , then ,
Edge - (ListM[i],ListM[i+1])
Add in ListM 2 - ((ListM[i],ListM[i+1]))

16. else if ListM[i] contains while , then ,
f=ListM[i]
if ListM[i+1] does not contain sp , then,
Edge(ListM[i],ListM[i]))
Add in ListM 2 - ((ListM[i],ListM[i]))
st=i
for j in range - i+1 to len(ListM)
{
if ListM[j] does not contain sp , then,
en=j
break
}

Edge - (ListM[i],ListM[j])
Add in ListM 2 - ((ListM[i],ListM[j]))
if(ListM[i+1]!=ListM[j]) , then ,
Edge - (ListM[i],ListM[i+1])
Add in ListM 2 - ((ListM[i],ListM[i+1]))

17. else if ListM[i] contains whilesp ,then ,
if(i>=st and i<=en) then
Edge(ListM[i],f))
Add in ListM 2 - ((ListM[i],f))
else:
Edge(ListM[i],ListM[i+1]))
Add in ListM 2 -
((ListM[i],ListM[i+1]
))

Edge(ListM[i],ListM[i]))
Add in ListM 2 - ((ListM[i],ListM[i]))

18. else if ListM[i] contains ifsp ,then ,
Add in ListM 3 - ListM[i], ListM[i], ListM[i+1] and ListM[i+2]
if(i>=stfor and i<=enfor):
Edge - (ListM[i],f1)
Add in ListM 2 - ((ListM[i],f1))
Add in ListM 3 - (ListM[i])
else if ListM[i-1] contains for , then ,
Edge - (ListM[i],ListM[i-1])
Add in ListM 2 ((ListM[i], ListM[i-1]))
else:
Edge - (ListM[i],ListM[i+2])
Add in ListM 2 - ((ListM[i], ListM[i+2]))
Edge - (ListM[i],ListM[i+1])
Add in ListM 2 - ((ListM[i], ListM[i+1]))

19. else if ListM[i] contains ifspexp , then ,
if(i>=st and i<=en) , then ,
Edge - (ListM[i],f)
Add in ListM 2 - ((ListM[i],f))
Add in ListM 3 - (ListM[i])
else if(i>=stfor and i<=enfor):
Edge - (ListM[i],f1)
Add in ListM 2 - ((ListM[i],f1))
Add in ListM 3 - (ListM[i])

20. else if ListM[i] constains elsesp ,then ,
if(i>=st and i<=en) , then ,
Edge - (ListM[i],f)
Add in ListM 2 - ((ListM[i],f))
Add in ListM 3 - (ListM[i])

21. i=i+1
22. }

23. if end in ListM 3 , then remove end from it.

24. For all nodes with frequency 1 in ListM 3 :
Edge – (node , end)
Add in ListM 2 – ((node, end)

This whole algorithm is used for creating the tool the will help in generation of control flow graph automatically. This control flow graph is used in path testing. Python is used here for development of this tool. Anyone pass a particular program here written in python. When this program is given as an input to this tool, the tool generate cross ponding CFG of this program.

IV. APPEARANCE OF TOOL

The front view of the tool is shown in figure 1. Which shows four portions a) Create CFG, b) Complexity, c) Find Paths and d) Paste Your Code Below.

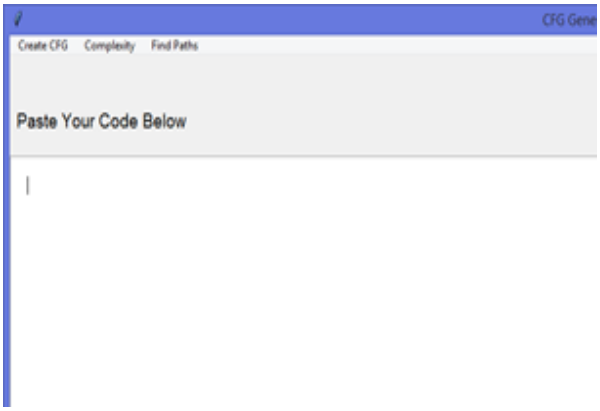

Figure 1: Front view of Tool

One example is given below with the use of this tool.

Input Program:-

```
var=int(input())
if(var == 200):
   print ("1 - Got a true expression value")
   print (var)
elif var == 150:
   print ("2 - Got a true expression value")
   print (var)
elif var == 100:
   print ("3 - Got a true expression value")
   print (var)
else:
print ("4 - Got a false expression value")
    print (var)
print ("Good bye!")
```

The above program is pasted in the area where the "paste your code below" is written and select Create CFG. As shown in figure 2.

After pasting the code go on Create CFG. When you click on Create CFG the Control flow of the particular program will be created as shown in figure 3.

Eg:-
List = ['start', 'if1', 'ifexp1', 'elif2', 'elifexp2', 'elif3', 'elifexp3', 'else4', 'end']

List 1 = [('start', 'if1'), ('if1', 'ifexp1'), ('if1', 'elif2'), ('elif2', 'elifexp2'), ('elif2', 'elif3'), ('elif3', 'elifexp3') , ('elif3', 'else4'), ('ifexp1', 'end'), ('elifexp2', 'end'), ('elifexp3', 'end'), ('else4', 'end')]

List 2 = {'if1': 2, 'ifexp1': 1, 'elif2': 3, 'elifexp2': 1, 'elif3': 3, 'elifexp3': 1, 'else4': 1}

Figure 2: Tool front view with pasted code

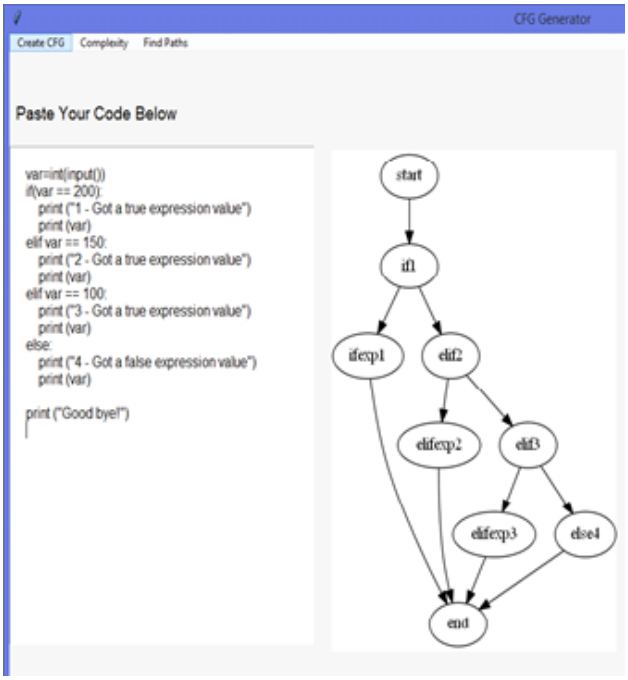

Figure 3: Tool Front View with control flow graph

V. COMPLEXITY

Complexity can be defined as the efficiency measure of any algorithm. Number of instruction execution for a logic written in terms of Algorithm is called its time complexity. Complexity of algorithm is less if time taken by the algorithm to execute is less and is considered as the good algorithm and if time taken to execute algorithm is more than it is not considered as good algorithm.

Complexity Calculated by the tool

This tool calculate the complexity of the program as shown in figure 4.

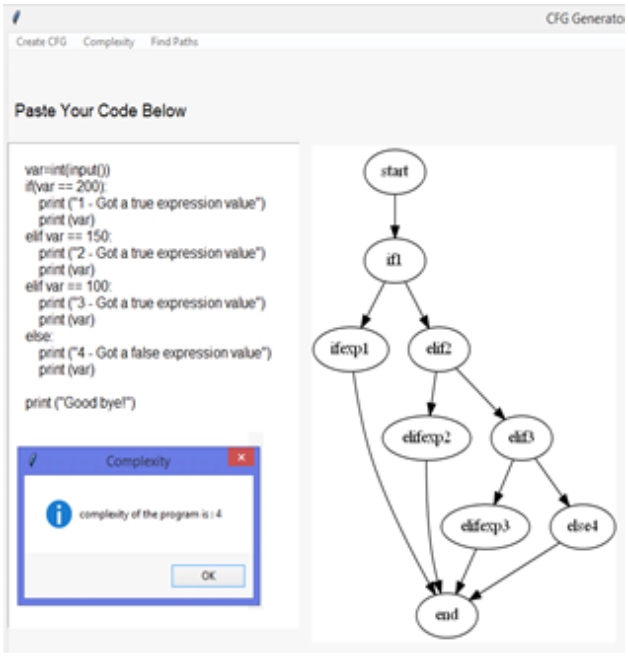

Figure 4: Tool Front View with complexity

Algorithm for finding different paths:-

Input:- List generated by CFG algorithm
Output:- List containing different paths

1. Create a dictionary (Suppose d).

2. for k,v in list 2: d.setdefault(k,[]).append(v)

3. Create a function (suppose find_all_paths)

4. Set source=’start’ and dest=’end’

5. find_all_paths (d, source, dest, path=[]) :

```
path = path + [source]
 if (source == dest)
        return [path]
   paths = []
for node in d[source]
   if node in path
      for i in d[node]
         if(i==node)
                break
         elif(i=='end')
                path=path+[node]+['end']
               paths.append(path)
               break
         else
              newpaths = find_all_paths(d, node,
              dest, path)
             for newpath in newpaths
                        paths.append(newpath)
                        return paths
```

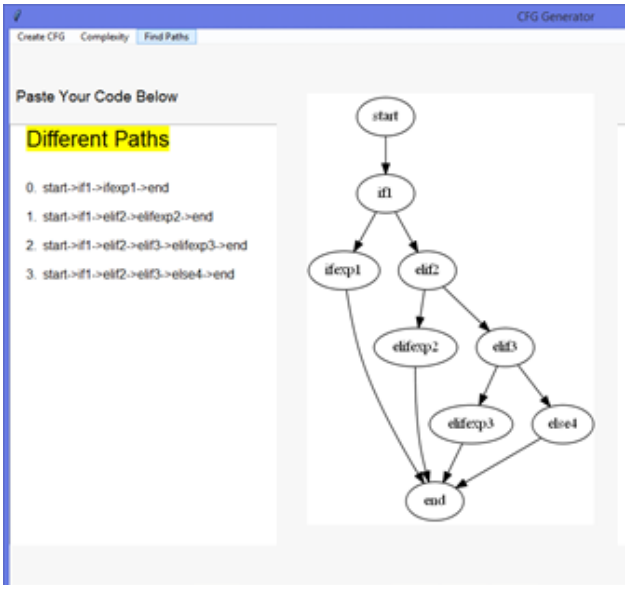

Figure 5: Tool Front View with different paths

Eg:-
Paths = [['start', 'if1', 'ifexp1', 'end'], ['start', 'if1', 'elif2', 'elifexp2', 'end'], ['start', 'if1', 'elif2', 'elif3', 'elifexp3', 'end'], ['start', 'if1', 'elif2', 'elif3', 'else4', 'end']]

VI. CONCLUSION

In this paper authors introduced a tool which takes a Python program file as input and generate a Control Flow Graph (CFG) for given Program. After generating the CFG it also find the all feasible paths of the CFG starting from first node and end at exit node. With the help of cyclomatic complexity all feasible paths have been found. This tool is used for Python programming language. Authors of this papers are also developed a tool which is working only for C programming [12].In the future scope automatic test cases can generate to cover with all feasible paths. For automatic generation of the test cases different nature inspired optimization techniques can be applied.

REFERENCES

1. Myers, G. J., Sandler, C., Badgett, T., 2011. The Art of Software Testing, 3rd Edition. Wiley Publishing.
2. Ammann, P., Offutt, J., 2008. Introduction to Software Testing, 1st Edition. Cambridge University Press, New York, NY, USA.
3. Patterson, A., K¨olling, M., Rosenberg, J., 2003. Introducing unit testing with BlueJ. In: Proceedings of the 8th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE 2003). ACM, pp. 11– 15.
4. Jones, E. L., 2001. Integrating testing into the curriculum – arsenic in small doses. In: Proceedings of the 32th SIGCSE Technical Symposium on Computer Science Education (SIGCSE 2001). ACM, pp. 337–341
5. Elbaum, S., Person, S., Dokulil, J., Jorde, M., 2007. Bug hunt: Making early software testing lessons engaging and affordable. In: Proceedings of the 29th International Conference on Software Engineering (ICSE 2007). IEEE, pp. 688–697
6. Astigarraga, T., Dow, E. M., Lara, C., Prewitt, R., Ward, M. R., 2010. The emerging role of software testing in curricula. In: Proceedings of the IEEE Transforming Engineering Education: Creating Interdisciplinary Skills for Complex Global Environments. IEEE, pp. 1–26
7. Clarke, P. J., Davis, D., King, T. M., Pava, J., Jones, E. L., Oct. 2014. Integrating testing into software engineering courses supported by a collaborative learning environment. ACM Transactions on Computing Education 14 (3), 18:1–18:33.
8. Wong, E., 2012. Improving the state of undergraduate software testing education. In: Proceedings of the 2012 Annual Conference of American Society for Engineering Education (ASEE 2012). IEEE, pp. 1–26.
9. Madeyski, L., Radyk, N., 2002. Judy: a mutation testing tool for Java. IET Software 4, 32–42.
10. Ramzi A. Haraty, Nashat Mansour & Hratch Zeitunlian (2018) Metaheuristic Algorithm for State-Based Software Testing, Applied Artificial Intelligence, 32:2, 197-213, DOI: 10.1080/08839514.2018.1451222.
11. Khan, R, Amjad M. "Optimize the software testing efficiency using genetic algorithm and mutation analysis." Computing for Sustainable Global Development (INDIACom), 2016 3rd International Conference on. IEEE, 2016.
12. Khan, R, Amjad M and Mishra M, Tool for Generating Control Flow Graph (CFG) Used in Unit Testing, Proceedings of the 11th INDIACom; INDIACom-2017; IEEE Conference ID: 40353 2017 4th International Conference on "Computing for Sustainable Global Development", 01st - 03rd March, 2017 pp 1161-1166.
13. T.J. McCabe, Structural Testing: A Software Testing Methodology Using the Cyclomatic Complexity Metric, NIST Special Publication 500-99, NIST, Washington, D.C., 1982.

A. Bagnall, V. Rayward-Smith, I. Whittley, The next release problem, Inf. Softw. Technol. 43 (14) (2001) 883–890.

14. M. Harman, S. Afshin Mansouri, Y. Zhang, Search-based Software Engineering: Trends, techniques and applications, ACM Comput. Surv. 45 (1) (November 2012), article no. 11.
15. J. Yan, J. Zhang, An efficient method to generate feasible paths for basis path testing, Inf. Process. Lett. 107 (3–4) (2008) 87–92.
16. Z. Zhonglin, M. Lingxia, An improved method of acquiring basis path for Software testing, in: Proceedings of 5th International Conference on Computer Science & Education, China, 2010, pp. 1891–1894.
17. D. Qingfeng, D. Xiao, An improved algorithm for basis path testing, in: Proceedings of the International Conference on Business Management and Electronic Information (BMEI), 2011, pp. 175–178.
18. J.R. Bint, Renate Site, Optimizing testing efficiency with error prone path identification and genetic algorithms, in: Proceedings 2004 Australian Software Engineering Conference (ASWEC'04), Australia, 2004, pp. 106–115.
19. K. Gallagher, M. Sambridge, Genetic algorithms: A powerful tool for large-scale nonlinear optimization problems, J. Comput. Geosci. 20 (7–8) (1994) 1229–1236.
20. Hermadi, Irman, Chris Lokan, and R. Sarker. "Dynamic stopping criteria for search-based test data generation for path testing." Information and Software Technology 56.4 (2014): 395-407.
21. Girgis, Moheb R., Ahmed S. Ghiduk, and Eman H. Abd-Elkawy. "Automatic Generation of Data Flow Test Paths using a Genetic Algorithm." International Journal of Computer Applications 89.12 (2014): 29-36.
22. Khan, Rijwan, and Mohd Amjad. "Automatic Generation of Test Cases for Data Flow Test Paths Using K-Means Clustering and Genetic Algorithm. "International Journal of Applied Engineering Research 11.1 (2016): 473-478.
23. Mahajan, Manish, Sumit Kumar, and Rabins Porwal. "Applying genetic algorithm to increase the efficiency of a data flow-based test data generation approach." ACM SIGSOFT Software Engineering Notes 37.5 (2012): 1-5.

AUTHORS PROFILE

First Author Mr. Akhilesh Kumar Srivastava is a Computer Science Graduate from K.N.I.T. Sultanpur and Post Graduate from Dr APJ Abdul Kalam Technical University Lucknow. He is registered for PhD at UPES Dehradun. Author is GATE and UGC Net Qualified. He has authored 4 Books and several research Papers in Software Testing and Wireless Sensor Network area in International/ National Journals, presented papers in International Conferences. Author runs his own You Tube Educational Channel with substantial viewership across the globe. Author has received several rewards from Academic institutes and Infosys Ltd.

Second Author Dr. Rijwan Khan is B.Tech (CSE), M.Tech (CSE), and PhD (CSE). Currently he is working as Head of Department CSE in ABESIT, Ghaziabad. He is author of more than 25 journal papers and 12 IEEE conference papers, one book chapter and 3 books. He has already published several research papers on software testing area.

Third Author Ms. Sneha Jain is a final year student of Computer Science & Engineering Department in ABES IT. She has carried out few projects in Sotware Testing domain.