Automated Testing Tool for Different Coverage Metrics

Shiva prasanth A[1] , Dhanalakshmi S[2],Shilpa S[3], kalyanisankar S[4],

student[1,3,4] Associate profesor[2]

Department of computer science and engineering

SNS College of Technology

Coimbatore

[1]shivaprasanthsp@gmail.com , [2]rajamdhana senthilkumar@gmail.com[3] shilpasundaram19@gmail.com ,

[4]sankar sekar@gmail.com

Abstract: Software testing is a critical element of software quality assurance and represents the ultimate review of specification, design and coding. It is an important activity carried out in order to improve the quality of the software. The main aim of this testing approach is to find the errors and to make the working of the software in efficient manner. Thus testing can be carried out by finding the test cases. A good test case is one that should have the high probability of finding the errors. There are different testing approaches and they are path coverage, branch coverage, code coverage.It is a popular approaches to measure the thoroughness of test suites. For each testing approach there are different tools available. In this paper we proposed the technique that are used to find the coverage metrics and common tool that can test all the coverage.

Key words: Branch coverage, Path coverage, Code coverage, Cyclomatic Complexity.

INTRODUCTION

Coverages are of different types and the approaches used for that also get varies. For each and every coverage metrices the execution methods varies. In this branch coverage can be found through symbolic execution and symbolic reachablity. This model identifies the frontier between symbolic execution[1][2][3] and symbolic reachability. It mainly focus on finding the rare execution conditions and eliminate the infeasible branches from coverage measurement.[4] It finds the product a modified branch metric that indicates the amount of feasible branches covered during testing. Generalized Control Flow Graph(GCFG)[5] that capture the state of analysis at each step.

Path Coverage is the structural testing strategy. It mainly focus on the flow of execution. This flow of execution can be founded by calculating cyclomatic complexity.[6] Flow graph is a graphical representation of logical control flow of the program. Such a graph consists of a circle called a flow graph node, which is known as node.[7] Through the help of flow graph the execution and the complexity of an program can founded. It mainly concerate on the cyclomatic flow in the graph. It also known as a cyclic flow graph. [8]Graph that consists of a cycles and acyclic nodes.

Code Coverage mainly focus on finding the code that has be executed while performing the test suites. There are different types in this code coverage such as statement coverage, loop testing,[11] condition testing, structural testing. Branch Coverage comes under this code coverage method.[10] It is very important metrices because each and every software are developed by executing the codes. Code coverage is used to test the loop constructs. Structural testing[11] is sometimes called white-box testing. Test cases is according to the program structure.

BIDIRECTIONALSYMBOLIC ANALYSIS

Bidirectional symbolic analysis coordinates the forward and backward analysis through the model.[1] It includes finding the reachability frontier, which guide the analysis. It includes finding the symbolic execution and symbolic reachability to improve and refine the branch coverage, by covering not-yet coverd branches and identifying the infeasible branches.[2] This analysis finds the target branches and drives the alternation of the different analysis steps. It includes finding the code that test case that not yet covered.

A.Symbolic Execution

Symbolic Execution targets the frontier edge and symbolically analysis towards the possible successor node. It mainly focus on the sequence flow of execution,[3][4] while we consider the control flow edge. This execution strategy includes drawing a graphic format of a given code. There are different diagrammatic approach for coding a control flow graph. All the possible flow of execution will be in this statement. There are three main computing process Symbolic execution, ComputeFrontierEdges, ModelCoarseningstep.

B.Symbolic Reachability

A symbolic reachability analysis step augments the GCFG with new reachability states.[5][4] It includes the sequence flow of a graph from top to bottom edges. Splitting of a flow graph into two equal parts and finding the execution is carried out through this execution strategy method. It produces a new state by augmenting the predicate with the information of the control flow branch.

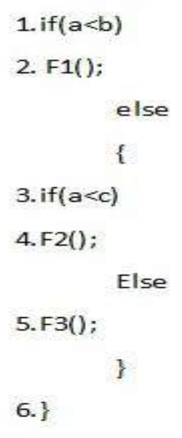
C.Frontier

The frontier states represent the best candiates for increasing branch [1][2][3]coverage by i) extending executed states towards reachability conditions that of uncovered branches ii) refining reachability conditions that cannot be satisfied from the currently executed states, iii) identified unsatisfiable reachability conditions to revel infeasible branches.[5] The identification of the frontier states prevents forward and backward analyses.

D.GCFG Model

The GCFG model[5] integrates the states computed with symbolic execution, the states computed with symbolic reachability analysis and the control flow relations among them.

The GCFG[5][4] represents the program branches executed during symbolic execution and identifies the target branches that are program branches that have not been executed and have not been identified as unreachable yet with symbolic reachability analysis



```
1. if(a<b)
2. F1();
          else
          {
3. if(a<c)
4. F2();
          Else
5. F3();
          }
6. }
```

Fig 1 Statement coding

The GCFG flow graph for above program can be represented by flow graph. The symbolic execution and the symbolic reachability can be founded with the help of given flow graph.

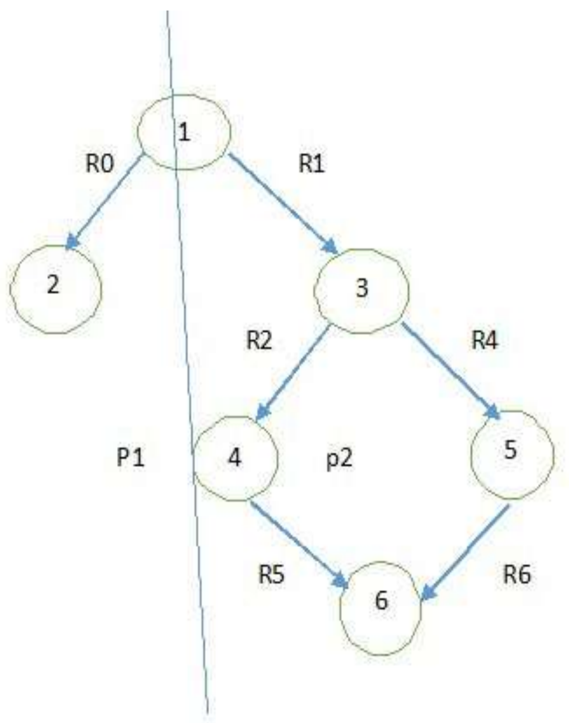


Fig 2 Control Flow Graph

Symbolic Execution : 1-R0-2, 1-R1-3, 3-R2-4, 3-R4-5, 4-R5-6, 5-R6-6

Symbolic Reachability: 1-R0-2,

1-R1-3-R2-4-R5-6,

1-R1-3-R4-5-R6-6

PATH COVERAGE

This coverage metrices is intended to exercise every independent execution path of the program at lease once. It is a important strategic approach in the testing metric.[7][8] There are sequence flow of steps that needs to carried out for maintaining a path coverage metric. The steps are as follows

1. Design the flow graph for the program or a component
2. Calculate cyclomatic complexity
3. Select a basis set of path[8]
4. Generate the test case for the paths

These are the steps that needs to carried out while performing the path coverage, Step 1 is a design of flow graph which is included in the fig no.2.2.[7][8] Cyclomatic Complexity can be calculated through the equation

$$V(G)= E-N+2$$

Where E-Number of edges in the flow graph

N- Total number of nodes in the flow graph

In the figure 2.2 the number of edges =7,
Number of Nodes=6
Cyclomatic Complexity=E-N+2
7-6+2
3
Complexity rate=3

Next includes finding the set of paths that are available in the flow graph. Paths are

Path 1: 1,2,6

Path 2: 1,3,4,6

Path 3: 1,3,5,6

Last steps includes finding the test cases. Test cases that included in this program are illustrated below

Test case 1: validating the list boundary and checking the loop condition statement

Test case 2: Checking[7][8] the second loop statement and checks whether it moves to the proper else statement.

CODE COVERAGE

Code Coverage[11][10] is a metric that includes finding the code that's are not executed while performing the execution of test cases. Coverage includes finding the which part of particular suite case runs. First method invented for systematic software testing. There are many tools for performing the code coverage metrices. Statement coverage comes under this technique.

if a and b then
Condition coverage can be satisfied by two tests
A=true b=false
A=false b=false

A. Modified Condition/Decision Coverage

It requires both decision and condition coverage be satisfied. Avionics software is required of modified condition/decision coverage(MC/DC).[10] It is a multiple coverage .It includes other types of coverage[9] such as loop coverage, entry/exit coverage, state coverage.

B.Loop Coverage

Loop testing is a white box testing technique which is used to test the loop constructs.[9] For simple loops there are different step concern,

i) n=0 that means skip the loop completely.

ii)n=1 that means one pass through the loop is tested.

iii)n=2 that means two passes through the loop is tested.

iv)n=m that means testing is done when there are m passes where m<n.

v)Perform the testing when number of passes are n-1,n,n+1.

Coverage Tools

There are different testing tools available in market, but depends on the tools coverage metrices[10]may get varies and some of the tools are given below

A.Jcov

Developed by sun JDK, it test from very beginning of Java,[11] distribution terms are GNU Public License, Version 2 with the class path Exception, open source, working with JDK.

B.JACOCO

Open source toolkit for measuring and reporting of Java Code Coverage, Eclipse Public License distribution terms. It is developed for replacement of EMMA. It instruments[10][11] bytecode, while running ECLEMMA Eclipse(software) is included.

C.Cobertura

Developed by stevan christou, it does not instrumenting the byte code, License GPL 2.0 Operating system cross-platform.

D.Emma

It is a latest release took place in mid-2005. As replacement of JACOCO works by wrapping each time of code and each condition with flag. It iss possible to dump[10] or result coverage data remotely without JUM exit. Outputs reports can highlight items with the coverage levels below thresholds.

E.Djunit

Djunit[10] is a eclipse plug in-that generated test coverage can be founded, Test with mock objects,and provide simple trace information about the cases.It is covering reporting tool.

F.Edumma

Edumma[11] is based on jacoco code coverage library. The eclipse integration has its focus on suppprorting the individual developer in high interactive way.

CONCLUSION

We conclude by saying that there are different tools available for performing different testing coverage. But the coverage level varies from tool to tool. So by having the common tool which can perform all the three coverage metrices the efficiency get increases. Through the proposed system only one common tool is used for perform the coverages. No need of going for separate tool for different coverages. Sequence flow is maintain by having the common automated testing tool for all coverage metrices.

REFERENCEES

[1] L. A. Clarke and D. S. Rosenblum, "A historical perspective on runtime assertion checking in software development," SIGSOFT Sooft. Eng. Notes,vol. 31, no. 3, pp. 25-37, May 2006.

[2] M.Baluda , P. Braione, G. Denaro,"Enhancing structural software coverage by increementally computing branch executability," Software quality journal,Volume 19,no.4,2011

[3] RTCA, Inc.,"DO-178C/ED-12C: Software considerations in airborne systems and equipment certification,"Dec 2011

[4] C.Cadar,D.Dunbar, "KLEE: Unassisted automatic generatoin of high-coverage tests for complex system programs," in proceedings of Eighth USENIX Symposium on operating systems design and implementation(OSDI 2008), 2008

[5] T. Xie, N. Tillmann, P. de Halleux, and W. Schulte,"Fitnessguided path exploration in dynamic symbolic execution,"in Proceedings of the 39th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2009), June-July 2009, pp. 359–368

[6] A. Bertolino and F. Basanieri. A practical approach to UML-based derivation of integration tests. Proceedings of the 4th International Software Quality Week Europe and International Internet Quality Week Europe, 2000.

[7] J.R. Birt and R. Sitte. Optimizing testing efficiency with error-prone path identification and genetic algorithms. In Software Engineering Conference, 2004. Proceedings. 2004 Australian, pages 106 – 115, 2004.

[8] Z. Zhonglin and M. Lingxia. An improved method of acquiring basis path for software testing. In Computer Science and Education (ICCSE), 2010 5th International Conference on, pages 1891 –1894, aug. 2010.

[9] C. Youngblut and B. Brykczynski, "An examination of selected commercial software testing tools", IDA Paper P- 2769, Inst. for Defense Analyses, Alexandria, Va., (1992) December.

[10] H. Zhu, P.A.V. Hall and J. H. R. May, "Software unit test coverage and adequacy", ACM Computing Surveys (CSUR), vol. 29, no. 4, (1997), pp. 366-427.

[11] M. Kessis, Y. Ledru and G. Vandome, "Experiences in Coverage Testing of a Java Middleware", In Fifth International Workshop on Software Engineering and Middleware (SEM'05), Lisbon, Portugal, ACM Press, (2005), pp. 39–45.