Minimal Test Case Generation for Effective Program Test using Control Structure Methods and Test Effectiveness Ratio

Swathi.J.N
Assistant Professor
SCSE, VIT University
Vellore

Sumaiya Thaseen.I
Assistant Professor
SITE, VIT University
Vellore

Sangeetha.S
VIT University
Vellore

ABSTRACT
Software testing is the critical activity in any industrial–strength software development process. As the software grows in size, its complexity increases and testing becomes more difficult. Hence generating test cases manually produces more errors and affects overall system quality. In this paper, we have proposed a tool for automatic generation of test cases using the control structure methods. This tool aims to achieve 100% coverage of a given structural code by including statement coverage, decision coverage, and path coverage and branch coverage analysis. It also helps the developers and testers to measure the effectiveness of test case generated using a metric called 'Test Effectiveness Ratio'.

General Terms
Software testing, Structural testing, Test case generation.

Keywords
Testing, Test cases, Control Flow Graph (CFG), Statement Coverage, Decision Coverage, Path Coverage, Branch Coverage, Test Effectiveness Ratio

1. INTRODUCTION
Software testing is defined as the process of executing a program with the intent of finding errors [6]. According to this definition, a good set of test cases are one that has a high chance of identifying previously unknown errors, while a successful test run is one that discovers these errors. In order to detect all possible errors within a program, exhaustive testing is required to exercise all possible input and logical execution paths. But in reality, as the software size grows its complexity increases and effective and exhaustive testing becomes impossible by human even for moderately complex systems. The minimum requirement of software testing would be to ensure correctness of the software system. Checking correctness of the system can be achieved by using a set of test cases which helps in identifying the errors.

Several heuristic methods are available for effective generation of test cases. The heuristics can be broadly divided into two categories: Black box testing and White box testing. Black box testing focuses on test data generated from functional requirements of the software which describes the behavior of the system without regard to the structure of the system [7]. Black box testing is also called as requirements based testing.

The other way of deriving test cases for black box testing is from formal specification. The formal specification is a mathematical description of the software that may be used for development of the product. It is widely used in practice as the specification would be precise and unambiguous [8]. The formal specifications are used in traditional as well as in object oriented approaches. The problem in using formal specification for generation of test cases is as the customer requirements change the specification changes and lot of code gets modified resulting in rework of test cases.

To alleviate this problem, test cases can be generated from the procedural design or program code which is also called structural testing or white box testing where the structure and the flow of the software is visible to the tester.

The paper is structured as described in the following: In Section 2 the concept of structural testing is discussed briefly. The proposed work is given in Section 3.In Section 4 the complete System Architecture and Detailed Design (Class Diagram) is presented. The Results and their Discussion can be found in section 5.Finally, Section 6 concludes with a short summary of the tool along with proposed future work.

2. STRUCTURAL TESTING

Structural testing takes into account internal structure of the program which in turn is divided into data flow and control flow criteria. Data flow criteria are based on the investigation of the ways in which values are associated with variables and how these associations can affect the execution of the program. A control flow criterion examines logical expression, which determine the branch and loop structure of the program.

Structural testing includes the following aspects of coverage while generating test cases [2].

- Statement coverage (SC): every statement in the program has been executed at least once. The advantage of the statement coverage is its ability to identify which blocks of code has not been executed.
- Decision coverage/Branch coverage (DC): every statement in the program has been executed at least once, and every decision in the program has taken all possible outcomes at least once.
- All Path coverage (PC): every possible route through a given part of the code has been executed at least once.
- Simple Path coverage (SPC): every possible route through a given part of the code has been executed in which no program part is executed more than necessary.
- Condition coverage (CC): every statement in the program has been executed at least once, and every condition in each decision has taken all possible outcomes at least once;
- Decision/condition coverage (D/CC): every statement in the program has been executed at least once, every decision in the program has taken all possible outcomes at least once, and every condition in each decision has taken all possible outcomes at least once;
- Multiple condition coverage (MCC): every statement in the program has been executed at least once, and all possible combinations of condition outcomes in each decision have been invoked at least once.

Basis path testing method enables us to derive a logical complexity measure (Cyclomatic complexity) of a procedural design or program code. The value computed for cyclomatic complexity is the number of independent paths in the basis set of program. An independent path is any path through the program that introduces at least one new set of processing statements or a new condition [14]. When stated in terms of a flow graph, an independent path must have atleast one edge that has not been traversed before the path is defined.

Cyclomatic complexity [2] also provides us with number of tests that must be conducted to ensure that all statements have been executed at least once. Complexity can be computed in any one of the three ways

- The number of regions of the flow graph corresponds to the cyclomatic complexity.
- Cyclomatic complexity V (G), for a flow graph, G, is defined as V (G) = E- N+2 , Where E is the number of flow graph edges and N is the number of flow graph nodes.
- Cyclomatic complexity V (G), for a flow graph, G, is also defined as V (G) = P + 1 Where P is the number of predicate nodes.

3. PROPOSED WORK

Test case generation is the process of identifying a set of test data which satisfies the selected test criteria. To make testing successful, a test case generation tool which adopts the basis path testing method is developed. The tool accepts the source code or a procedural design as input. The Source code is considered as a foundation component and corresponding control flow formula is generated based on the total number of statements and the predicates in the structural code. The formula produces a value which gives the minimum number of test cases for each code coverage type namely statement path, simple path, all path and branch path coverage. Based on the count value, test cases are generated in such a manner that will force execution of each path in the basis set. The effectiveness of the test cases generated is measured by determining the test effectiveness ratio (TER) [13].

$$TER = \frac{\text{Number of statements exercised by the test case}}{\text{Total Number of statements in the source code}}$$

4. SYSTEM DESIGN

The proposed system architecture Figure 4.1 and the detailed design Figure 4.2 in view of classes and their responsibilities is given below:

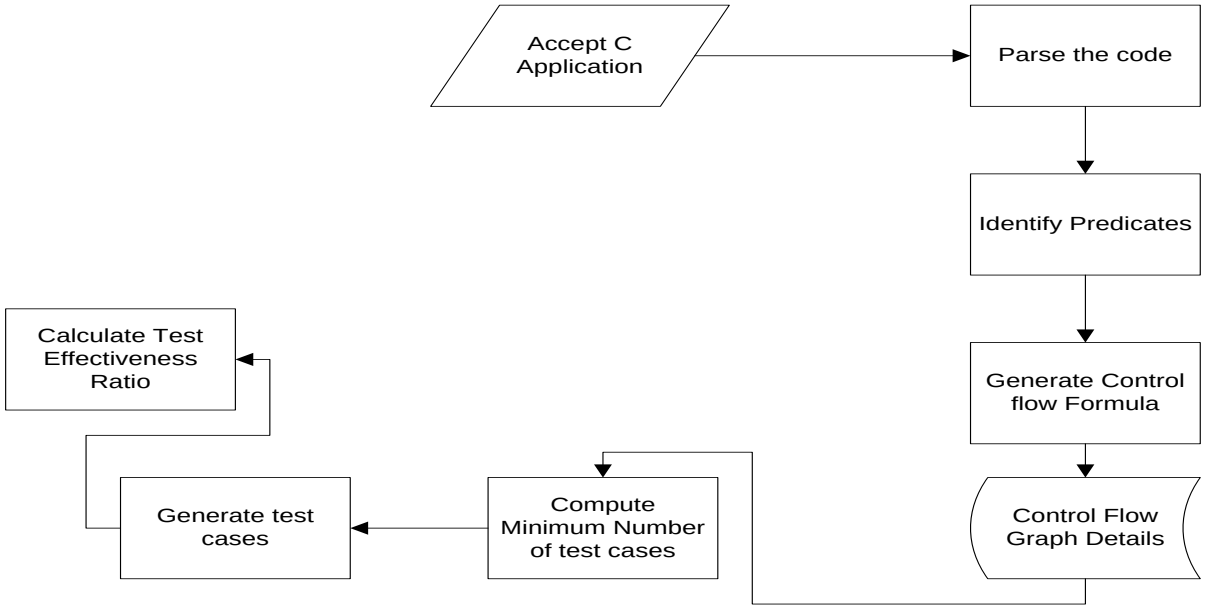

Figure 4.1 System Architecture

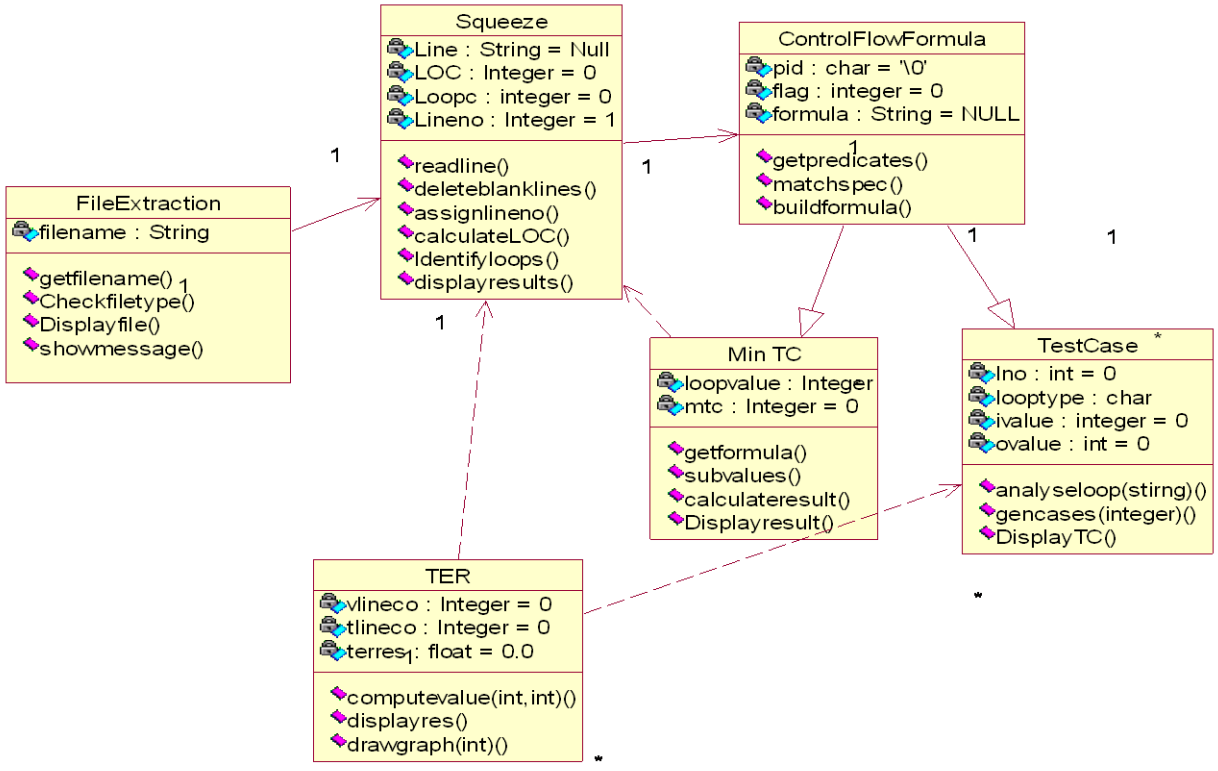

Figure 4.2 Detailed Design- Class Diagram

5. RESULTS & DISCUSSIONS

Figure 5.1 User Interface to view the Test Cases and Test Effectiveness Ratio-Statement Coverage

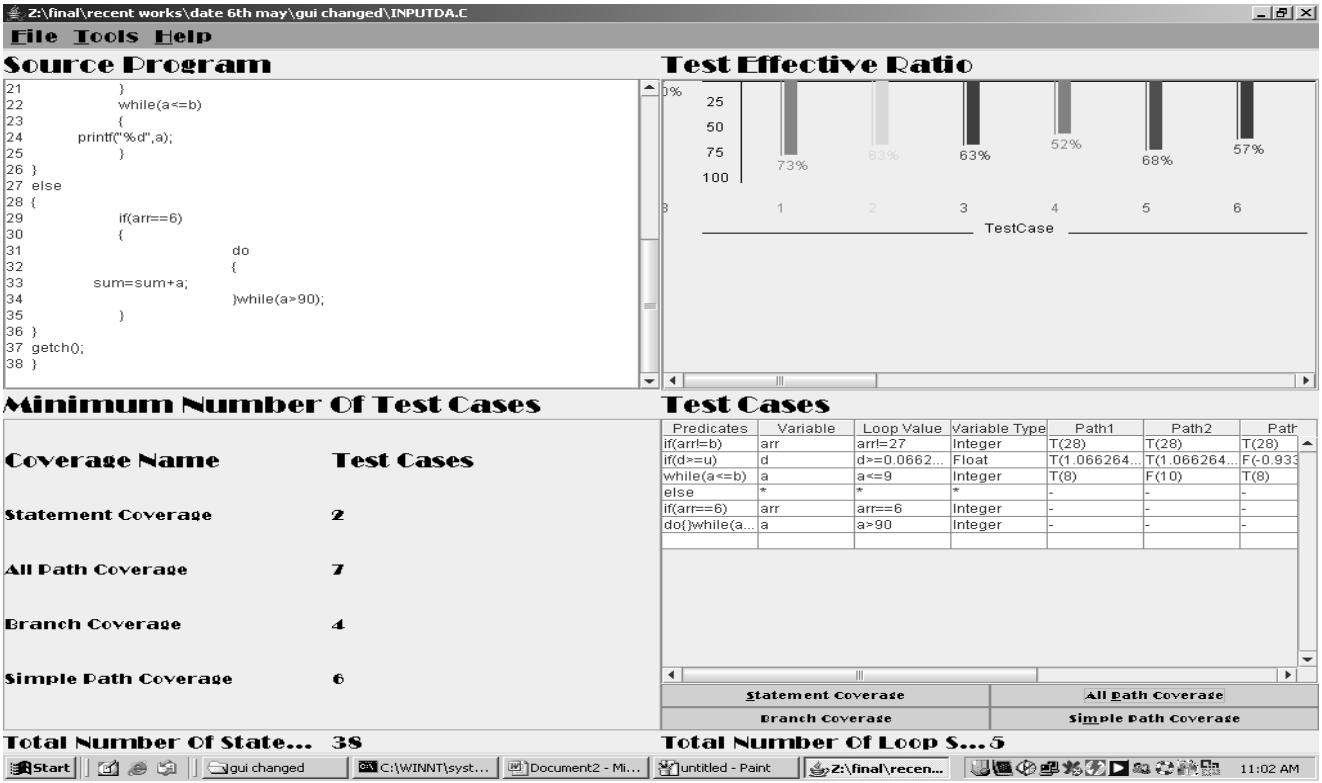

Figure 5.2 User Interface to view the Test Cases and Test Effectiveness Ratio-All path Coverage

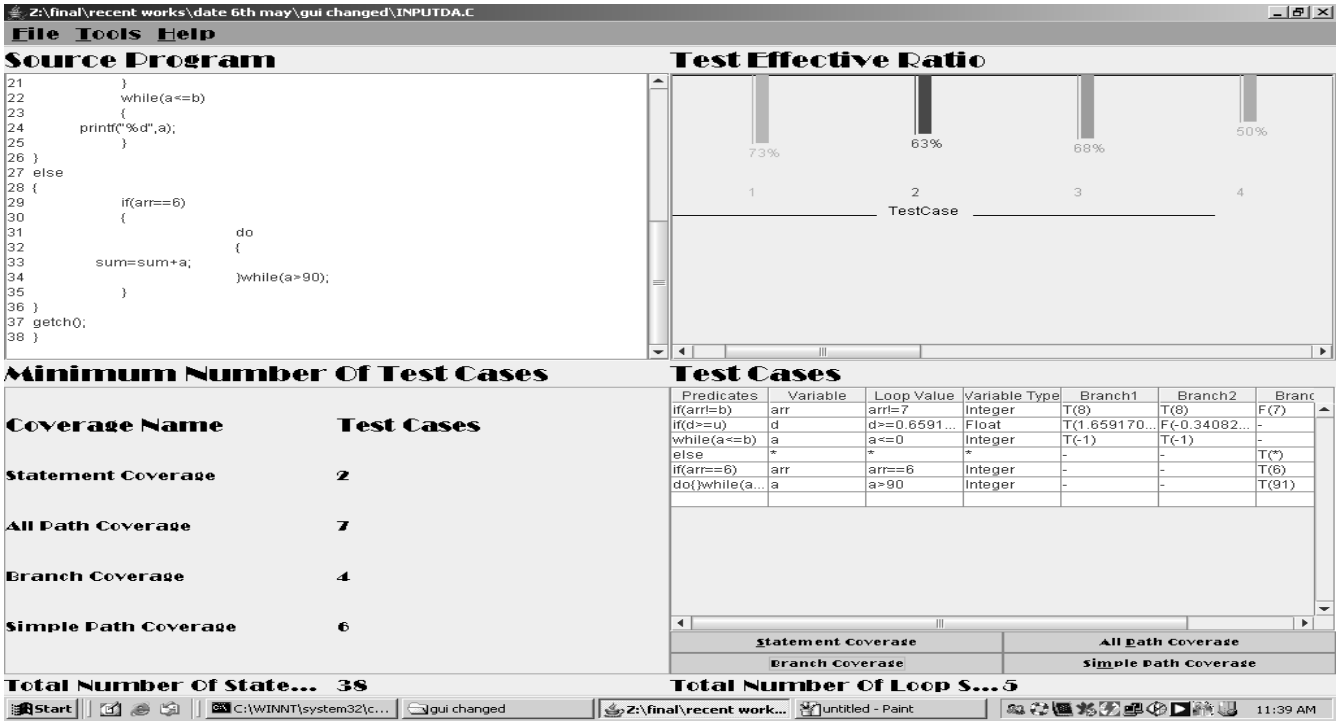

Figure 5.3 User Interface to view the Test Cases and Test Effectiveness Ratio-Branch Coverage

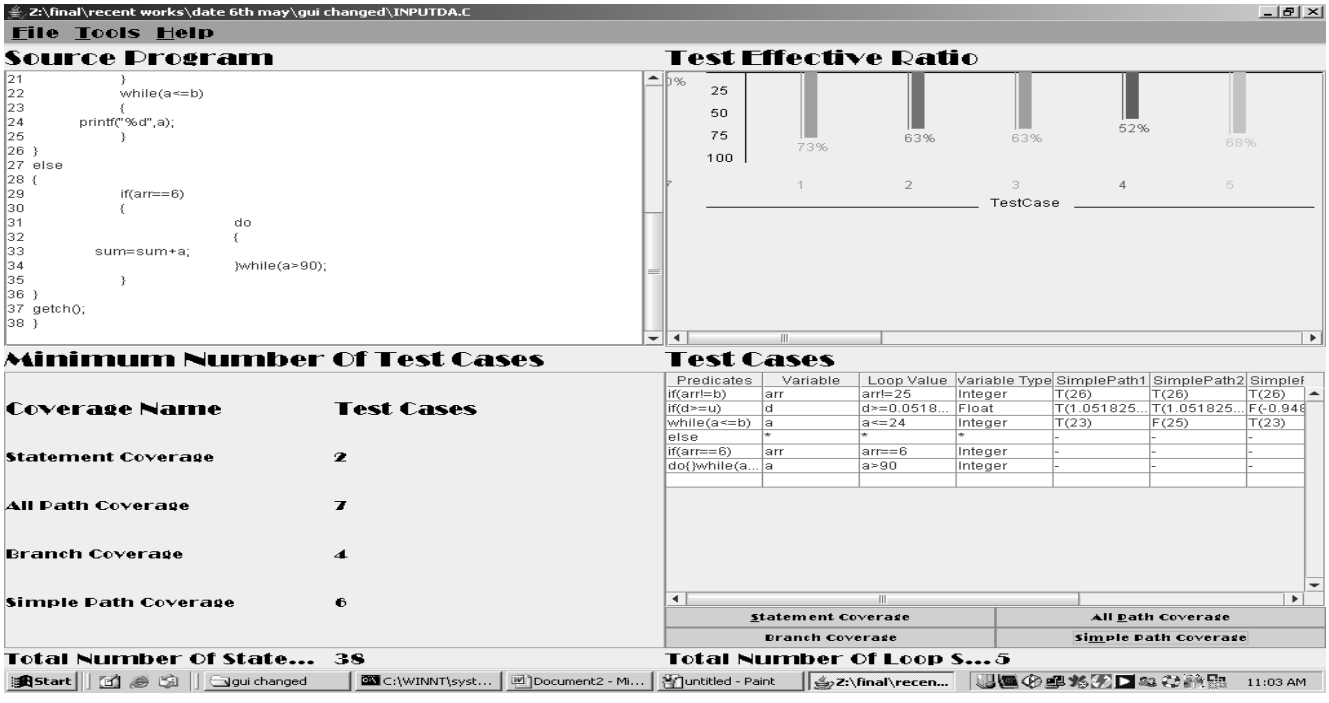

Figure 5.4 User Interface to view the Test Cases and Test Effectiveness Ratio-Simple Path Coverage

Generating Minimal Test Cases for effective program test method first identifies a set of paths in the program's flow graph, which covers all branches, paths and statements. It guarantees that all independent paths within the module have been exercised at least once, exercised all logical decisions on their true and false sides, and execute all loops at their boundaries and within their operational bounds. Then, it identifies the test data such that every selected path is executed. In this approach, test case exercises every branch in a program with numeric input, arrays, assignments, conditionals and loops.

Comparative Study:

The tabu search algorithm [7] use the control flow graph, which stores relevant information (for example the best tests and their costs). The goal is to automatically obtain branch coverage. In scatter search [16] test case generator uses the control flow graph in order to determine the covered branches. Each node has a solution set and the algorithm will try to make the sets as diverse as possible, using a diversity function to generate solutions that can cover different branches of the program, whereas the proposed tool uses the control flow graph to generate test case and test data in order to cover all branches, statements, paths and decisions by visiting all the nodes and the edges.

6. CONCLUSION

The proposed tool has been designed and developed after a detailed investigation of the 'C' code. The outcome of this tool is to assist the tester to test the code in efficient manner. The generated test cases will cover the code with maximum extent which is the major criterion for testing. The Test Effectiveness Ratio helps the tester in measuring the effectiveness of the test cases. The future work would be to generate test cases by including other code coverage analysis techniques like Condition coverage, Multiple Condition coverage etc. Another future work would be to trace the control flows between the function calls and to generate the test cases for the same.

7. REFERENCES

[1] Andreas S. Andreou, "An Automatic test-data generation scheme based on data flow criteria and genetic algorithms", Third International Conference on Natural Computation (ICNC) Vol. 1, pp. 2, 2007.

[2] A. H. Watson and T. J. McCabe, "Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric", Computer Systems Laboratory, National Institute of Standards and Technology, September 1996. NIST Special Publication 500-235.

[3] Beizer Boris, "Software Testing Techniques", 2nd edition, New York: Van Nostrand Reinhold, 1990.

[4] Carlos Urias Munoz,"An approach to software Product testing", IEEE Transactions on Software Engineering, Vol 14, No.11,pp.1589-1596, November 1988.

[5] Christophe Paoli, Marie Laure Nivet, Jean Francosis Santucci, Antoine Campana, "Path Oriented Test Data Generation of Behavioral VHDL Description", IEEE Transactions on Software Engineering, March 2002

[6] G. J. Myers, "The Art of Software Testing", John Wiley & Sons, Inc., Second edition, New York, July2004.

[7] Geetha Devasena M.S. and Valarmathi M.L., "Optimized test suite generation using tabu search technique", International Journal of Computational Intelligence Techniques, ISSN: 0976–0466, Vol 1, Issue 2, pp.10-14, 2010.

[8] Hung Tran, "Test Generation using Model Checking", Technical Report.

[9] Harman M, Hu, L, Hierons R., Wegener, J. Sthamer, H, Baresel, A & Roper, M, "Testability transformation", IEEE Transactions on Software Engineering, Vol 30, Issue 1, pp.3-16, Jan.2004

[10] Korel B,"Automated Software Test Data Generation" IEEE Transactions on Software Engineering, Vol 16, Issue 8, pp.870-879, Aug.1990.

[11] Michael C.C, McGraw G, schatz. M.A,"Generating software test data by evolution", IEEE Transactions on Software Engineering, Vol 27, Issue 12, pp. 1085-1110, Dec.2001.

[12] McCabe, Tom, "A Software Complexity Measure", IEEE Transactions on Software Engineering., Vol.2, No.6, pp.308-320, December 1976.

[13] M.R.Woodward, D.Hedley and M.A.henell, "Experience with path analysis and testing of programs", IEEE Transactions on Software Engineering", Vol 6, No.3, pp.278-286, 1980.

[14] Peres L.M., Vergilio S.R, Jino M and Maldonado J.C. "Path selection in the Structural Testing: Proposition, Implementation and Application of Strategies", XXI International Conference of the Chilean Computer Science Society, pp.0240, 2001

[15] Patricia Mouy, Marre B, Williams N and Le Gall,"Generation Of All Paths Unit Test with Function Calls ", 1st International Conference on Software Testing, Verification and Validation, pp 32-41, April 2008.

[16] Raquel Blanco, Javier Tuya, Belarmino Adenso Diaz, " Automated test data generation using a scatter search approach", Journal of Information and software Technology, Vol.51,Issue 54, April 2009.

[17] Sangeeta and Dr. Dharmender Kumar, "Automatic Test Case Generation of C Program Using CFG" , IJCSI International Journal of Computer Science Issues, Vol. 7, Issue 4, No 8, July 2010

[18] W. E. Wong, Y. Lei, and X. Ma.", Effective Generation of Test Sequences for Structural Testing of Concurrent Programs ", ICECCS Proceedings of 10th IEEE International Conference on Engineering of Complex Computer Systems, pp. 539–548, 2005.