International Journal of Technical Innovation in Modern Engineering & Science (IJTIMES)
Impact Factor: 5.22 (SJIF-2017), e-ISSN: 2455-2585
Volume 4, Issue 11, November-2018

COVERAGE BASED TEST_CASE GENERATION USING AUTOMATED TEST CASE GENERATING TOOL

Bhumikaben Patel, Jignesh Patoliya
(PG Student) (Assistant Professor)

V.T.Patel Department of Electronics & Communication,
Chandubhai S. Patel Institute of Technology (CSPIT),
Charotar University of Science and Technology (CHARUSAT)

ABSTRACT:-This paper presents a method for automatically generating test cases for different code coverage criteria of Function coverage, Statement Coverage, Brach Coverage and Condition Coverage to measure what percentage of code has been exercised by a test suite, Coverage criteria are usually defined as rules or requirements, which a test suite needs to satisfy.We show that how the Automated Test Case Generating Tool can be used to automatically generate complete test scenarios that will provide percentage coverage of test suit.
The purpose of producing the tool to help reduce the high cost of developing test cases for safety-critical avionics applications as well as to well as to save the time of deriving test cases manually.

INTRODUCTION

Software development for critical avionics control systems, such as the software controlling aeronautics applications like Aircraft Flight Control System, Engine Control System, Flight Management Control Systems are costly, time consuming, and error prone process. In such projects, the validation and verification phase (V&V) consume approximately 50%-70% of the software development resources. Thus, if the process of deriving test cases for V&V could be automated and provide requirements-based and code-based test suites that satisfy the most stringent standards (such as DO- 178C-the standard governing the development of flight-critical software for civil aviation , dramatic time and cost savings would be realized.[1]

This paper presents a method for automatically generating test cases for Statement Coverage, Brach Coverage and Condition Coverage of test suit. We show how a tool can be used to generate complete test cases that provide a predefined coverage of any software development artifact.

We provide a tool that is
(1) suitable for defining our test-case generation approach[1]
(2) defining how Statement Coverage, Brach Coverage and Condition Coverage criteria can be formalized in our framework and expressed

Random Testing Technique is used in this automatic test case generation tool which select values randomly from the input domain of the program input variables.[6]

SCOPE
Test case and Test conditions generation of (switch case, if-else, while loop and for loop with different operators) of test suit.

OUT OF SCOP
This tool will not take any other files except .C files as an input. So it cannot be used for testing any other source code file. It will not consider any other operations of C code out of these conditional statements.

STRUCTURAL COVERAGE ANALYSIS (SCA)
In software testing structural coverage analysis is one of the most important part and here we are going to use this for generate test cases. Each requirement in application one or more tests which prove that it has been implemented correctly. Structural coverage proves that these tests exercise all of the code. According to DO-178B Structural coverage will be statement coverage, decision coverage and MC/DC coverage depending on the software level. DO-178B (like requirement based testing), which perform structural coverage has some key benefits: requirements are complete with respect to code, test cases are complete, no code is deployed that shouldn't be there, code for use in other configurations is clearly identified.
Structural coverage includes: Statement coverage, Decision coverage, MC/DC coverage.
Statement coverage measures whether each statement encountered. This is affected by computational statements than by decisions.

For example:

```
If((x>1)&&(y=0))
{
z=z/x;
}
If((z=2)||(y>1))
{
z=z+1;
}
```

By x=2, y=0, z=4 as input every statement is executed once.
Statement coverage also include condition coverage, multiple condition coverage, loop coverage.
Decision coverage measures whether Boolean expressions such as if statement and while statement evaluated to both true and false.
For example:

```
If(a>b)
{
Print("hello");
}
Else
{
Print("bye");
}
```

Here either true case or false case so both true and false test case will be generated.
This Structural Analysis is method which will analyze the whole code of the software covering the conditions, loops branches and statements and accordingly it will generate appropriate test cases automatically which do not need human interaction for check.
This analysis will also check for the Standards an avionics software should follow which will consider for validity to get FAA Certificate(for verifying proper working of the software for aircraft systems).

FLOW OF THE SYSTEM

Following figure shows steps of the system to generate the Test Cases from .C file. System Parse the .C file using Parser and Lexer. Parsed data are used to extract the conditional and looping statements present in .C file. It is finally used to get the variable and different data values from .C file which is used to generate the Test Cases according to the different coverage based on the Avionics systems.

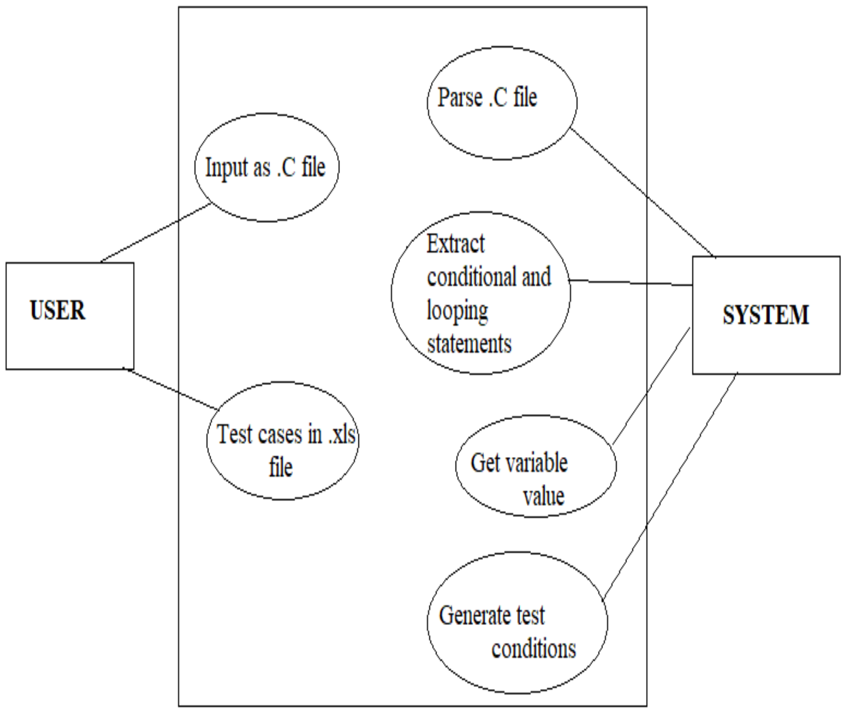

Fig1. Flow Diagram of System

WORKING OF THE SYSTEM

Input Data:

Data shall be C source code file or male directory. Preprocessed file shall also be accepted for the procedure. Input shall be taken by user by choosing path where the input file is situated. User shall also select whole directory where number of source code files are situated.

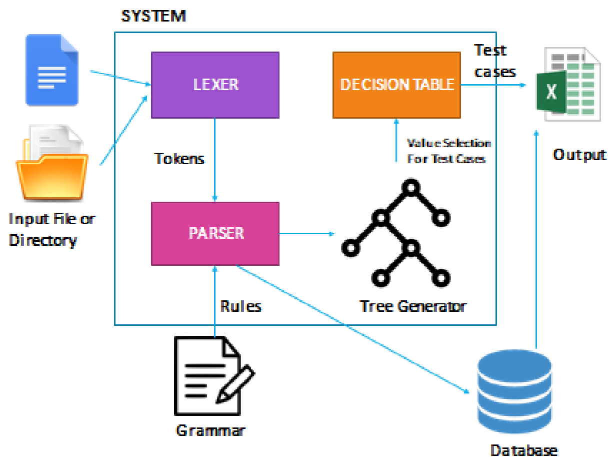

Fig2. Block Diagram of the System

Lexer:

This process will perform lexical analysis on the source code i.e input.
It will divide the code into tokens and store in database. These tokens can be identifier, keyword, separator, operator, literal, or comment. Examples of tokens are listed below:

Token name	Sample values
Identifier	Val1,y,var
Keyword	If , while , return
Separator	{ , (, ;
Operator	+ , > , =
Literal	False , 3.02e37 , “name”
Comment	/*function to add numbers*/ , //display output

Table1. Types of Tokens

Parser:

Parser is used as a compiler. Lexer is also a part of Parser. Parser considers the tokens as input, it will evaluate the conditional expressions using different stacks and grammar given to it.

Generate Tree (AST):

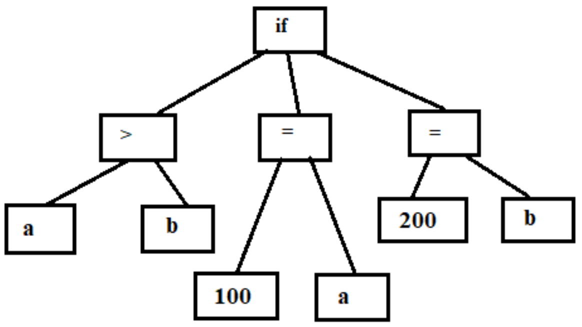

Fig3. Example of AST

Now the Abstract Syntax Tree will be generated according to grammar. It will represent structure of source code written in C programming language.
It would contain variables and those will be consider for decision table. These variables given automatically generated values and using those values output will be generated for test case.

Output Data:

The decision table will store all the variables and its values for generated test cases covering all the conditions (True-True, True-False, False-True, False-False) in .xls file. Output will be shown in this .xls file, where input file name, function name, line number, conditional statement, test cases related to that statement and outputs generated according to those test cases are listed.

CONCLUSION AND FUTURE WORK

We have demonstrated an approach of automate the test-case generation for avionics software engineering artifacts of source code. We use different coverage criteria to define what test cases are needed and the test cases are then generated using the automated test-case generation tool.

Results of the tool indicate that the approach has potential to dramatically reduce the costs associated with generating test-cases to high levels of coverage.

Future scope includes the Automated Test Case generation tool required a certain level of coverage MC/DC (Modified Condition Decision Coverage) that needs to be added for avionics systems DO-178C level A V&V.

REFERENCES

[1]. Sanjai Rayadurgam and Mats P. E. Heimdahl. **Coverage Based Test Case Generation using Model Checkers.** Department of Computer Science and Engineering, University of Minnesota.

[2]. Online resource: Wikipedia

[3]. Pankaj Jalote and Mallaku G. Caballero. **Automated Testcase Generation for Data Abstraction.** Department of Computer Science and Institute for Advanced Computer Studies, University of Marylend.

[4]. Online resource: aerointerview.com

[5]. Online resource: rapitasystems.com

[6]. Sangeeta Tanwer and Dr. Dharmender Kumar. **Automatic Testcase Generation of C Program Using CFG.** Computer Science and Engineering, Guru Jambheshawar University of Science and Technology Hisar, Haryana, India.

[7]. Kamran Ghani and John A. Clark. **Automatic Test Data Generation for Multiple Condition and MCDC Coverage.** Department of Computer Science, University of York, York, YO10 5DD, UK