Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X

IJCSMC, Vol. 4, Issue. 4, April 2015, pg.259 – 264

REVIEW ARTICLE

A Review Of Mutation Testing Architecture

Neetu Rani[1], Jyoti Chaudhary[2]
neeturathee22@gmail.com, jyotiseptember20@gmail.com
M.Tech Scholar, TITS[1]
Bhiwani, Haryana
Assistent Professor, TITS[1]
Bhiwani, Haryana

Abstract- Mutation Testing is white box testing technique that analyze the code under different values, bounds and constraints. It also analyzes the data values associated to different operators and evaluates their significance over the code. Further this operator based analysis helps in identifying the scope and limitations along with the relevancy, robustness and effectiveness of code. Mutation testing is applicable for all kind of procedural and object oriented operators. In this paper some aspects of mutation testing are identified respective to different associated constraints and limitation. The architectural analysis along with program flow analysis is mainly discussed and presented in this paper.
Keywords: Mutation Testing, Operator Specific, Procedural, Object Oriented

I. INTRODUCTION

White Box testing is also called open box testing that analyze the program or the program structure in different ways. This kind of analysis includes the program flow analysis, loop analysis, unreachable statement identifications, scope of variables, functional analysis, interface anlaysis etc. One of such white box testing approach is used to analyze the aspects respective to associated variables and the operators. According to this analysis, the program flow is identified in terms of different operators and expressions used in the program code. Once these operators and expressions are identified, the next work is to analyze the robustness of these operators and expression. This code level anlaysis is about to evaluate the code level integrity. This integrity itself represents the reliability of the software system with specification of operator and data integration. It also identifies the validity of the associated values. The analysis stages are shown in figure 1.

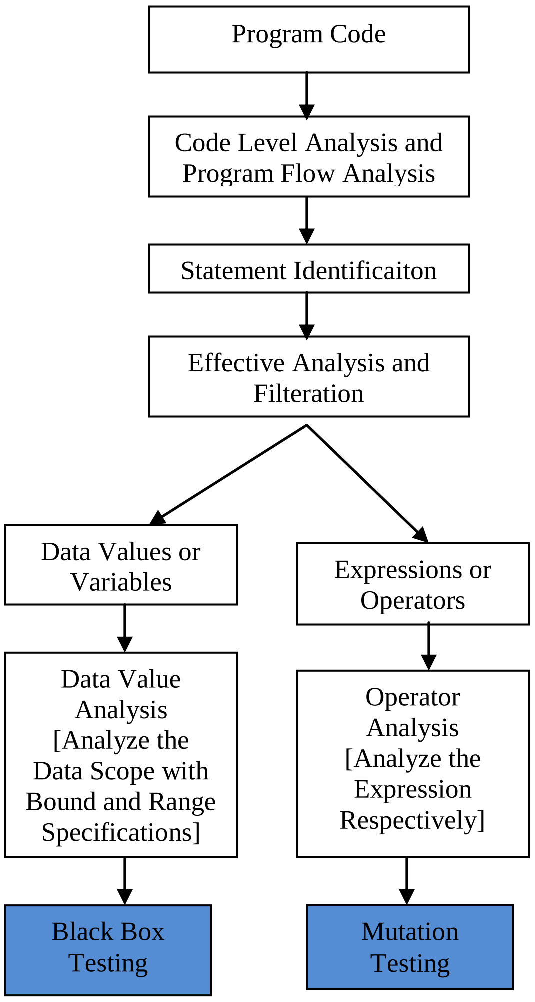
Program Code
Code Level Analysis and Program Flow Analysis
Statement Identificaiton
Effective Analysis and Filteration
Data Values or Variables
Expressions or Operators
Data Value Analysis [Analyze the Data Scope with Bound and Range Specifications]
Operator Analysis [Analyze the Expression Respectively]
Black Box Testing
Mutation Testing

Figure 1 :Mutatoin Testing Framework

Here figure 1 is showing the basic framework and work stages for mutation testing. The work of mutation testing begins while accepting the program code as input. This input code is then analyzed respective to the program flow analysis. This flow analysis is defined to identify the type of statements used in the program such as conditional statements, loops etc. It will also identify the infinite loop cases and the unreachable stages. While performing the effective analysis and filteration only the valid program statements level. Over these statements, only the statement with some expression or operator involvement are considered for mutation testing. This kind of statements can be obtained by eliminating the input/output statement over the program code. After obtaining the actual operator statements or the expression statement. The second level anlaysis is performed.

This second level analysis can be performed respective to the variable or the data values as well as respective to the operators. The data value specific analysis comes under blackbox testing in which the zero value analysis, negative value analysis and the boundaries values analysis is performed. Different test cases are performed at run time to perform this kind of analysis. To perform the mutation testing, the expressions are analyzed respective to different operators. These operators includes airthmetic operators, conditional operators, comparison operators, logical operators, object oriented operators etc.

The operators specific mutation testing analysis can be done in two main stages. In first stage, the identification of the operator class is done and the acceptance of the operator to the expression is applied. As an operator is applied or analyzed for a particular expression, it is called a Mutant. The mutant that provide the positive result is called alive mutant and that does not provide the valid result is called dead mutant.

The paper analysis the various aspects of mutation testing along with architectural specification. In this section, the basic structure of the mutation testing is defined. The section described the significance of mutation testing along with the specification of work stages. Section II, elaborates the work done by earlier researchers on mutation testing. Section III, the architectural analysis is discussed. Section IV, gives the conclusion of the study.

II. LITERATURE REVIEW

Lot of work is already defined by different researchers on mutation testing. This work defined under architectural differences, extraction approaches and the language implied. The number of operators and type of queries handled by different researchers are also different. Some of these architectural specification done by different researchers in mutation testing are described in this section:

John A. Clark[1] has defined a semantic analysis based mutation testing approach. Author provided the descriptive analysis on the program code and generate the language semantics. Each language semantic is here defined as the separate mutant. These mutants are captured under the fault level analysis. The context specification and dependency analysis is here been performed to generate the semantic mutants. Author applied the work for C programs and obtain the reliable and effective testing over the code.

Mike Papadakis[2] has defined two level mutation testing for empirical evaluation of code. Author defined this work in these two stages. In first stage, the application oriented cost analysis is performed and the equivalence testing is generated over the code. In second stage, the effective test evaluation is performed based on the strategy analysis. Author defined the mutation generation and its validation analysis so that the reliability of the code will be analyzed. Author defined the work respective to the quality estimation of program code under mutation testing.

Alper Sen[3] defined coverage analysis based concurrent analysis scheme to analyze the C code under different operators. Author analyzed the program code under operator specification and concurrency construct analysis. Author identified the unique operators over the program code and obtain the concurrent coverage metric so that the multiple execution schedules will be generated over the code. Author anlayze the program code under adequately measure on coverage and the concurrent programs will be analyzed. Author performed the experiments with different program designs and identify the scope of the program respective to operators and data specifications.

Fevzi Belli[4] defined a formal framework to optimize the mutation testing. Author defined a grammer specific analysis defined on the mutation testing under different mutation operators. Author defined the coverage respective to the concept and algorithm and provided the test sequence generation to optimize the relevancy of work. Author defined the framework to generate the model specific test cases and provided the dependency analysis under grammer evaluation. Author defined the preferences and case study to improve the data validation and analyze the characteristics issues over the program code.

K. K. Mishra[5] has defined an elitist genetic algorithmic approach to optimize the mutation testing. Author defined a fault based analysis to reduce the expensive process and reduce the criticality of the program. The evolutionary algorithmic here defined to reduce the cost of program analysis and to identify the effective blocks where test cases will be applied. Author identifies the associated faults while generating the mutants and kills the non required mutants at early stage so that the reliability of the approach is improved. Author presented the approach to generate the effective test cases with input data in context with mutation testing.

Lingming Zhang[6] has defined dynamic test case generation framework under symbolic execution of mutants. Author defined the preanalysis approach on the code to generate the dynamic symbolic execution and reducing the effect of meta program. Author provided self killing constraints that analyze the code and filter it dynamically while processing the code. Author provided the guided approach to improve the mutation approach.

Mohsen Fallah Rad[7] define a genetic improved bacterial analysis approach to optimize the mutation test process. Author reduced the overall cost by improving the test data and evolutionary process. Author provided the algorithmic approach to simulate the work and compare it with other algorithmic approaches to generate the mutant effective. Author reduces the death rate of mutants so that the reliability of system is improved.

Marinos Kintis[8] defined an evaluation effective mutation testing approach under test introduction, assessment and comparision. Author analyzed the system under various effort estimation and its reduction with identification of associated risk. Author identify the restrictions at early stage so that more accurate and effective mapping to the operators and expressions is done.

Panya Boonyakulsrirung[9] has presented weak testing framework to optimize the mutation process. Author defined the mutation generation and deletion at the early stage. Author provided the execution time analysis, mutation score anlaysis so that the effectiveness of work is improved.

III. MUATION TESTING ARCHITECTURE

In this section, an architectural discussion on the mutation testing is defined. The mutation testing framework basically depends on the specification of mutant generation over the program code. To perform this kind of analysis, the program code will be verified respective to different aspects so that the derivation of the program with code segment generation and operator identification. This kind of analysis can be defined under following constraints:

- The first work is to divide the program in statement types and identification of these statement blocks will be done. These blocks are defined under the language specification and language keywords specifications.

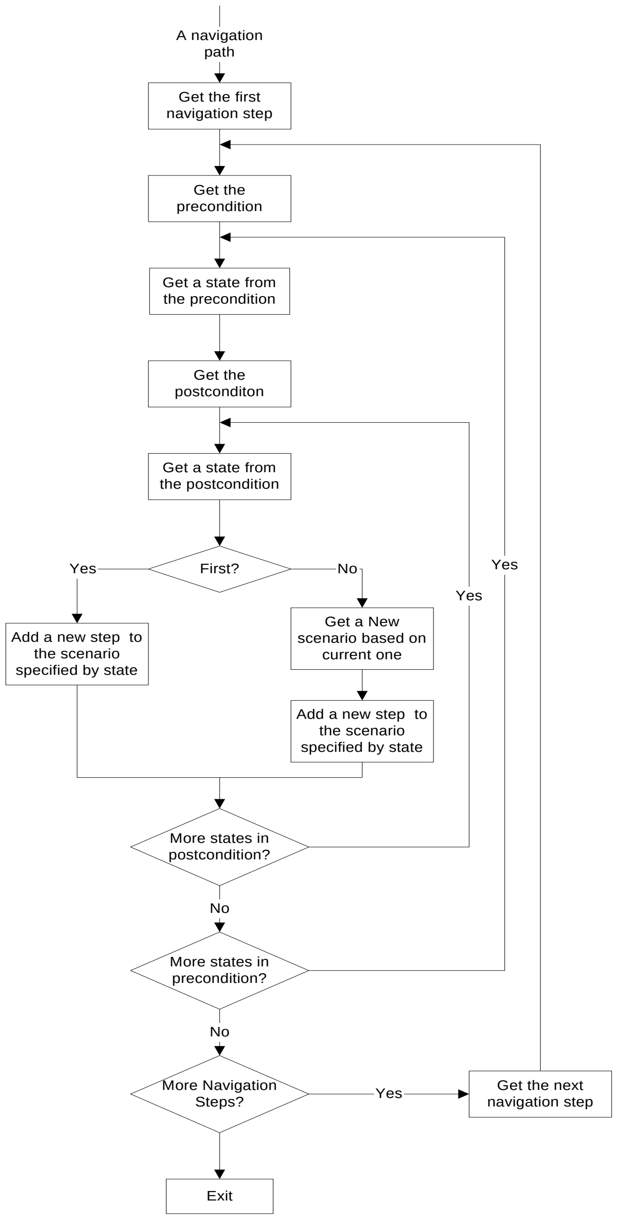
A navigation path
Get the first navigation step
Get the precondition
Get a state from the precondition
Get the postconditon
Get a state from the postcondition
First?
Yes
No
Add a new step to the scenario specified by state
Get a New scenario based on current one
Add a new step to the scenario specified by state
Yes
Yes
More states in postcondition?
No
More states in precondition?
No
More Navigation Steps?
Yes
Get the next navigation step
Exit

Figure 2 : Mutation specific code analysis framework

- These statements includes input-output statement, conditional statement, functional statement, loops etc.
- The identification of the expression in various statements is done. The statement that does not involve any expression will be eliminated from the code.
- Number of operators used in the expression are identified.
- Type of operators and operator class is identified.
- Identification of other operators from the same class are identified.
- The identification of operators that can be replaced to optimize the code can be done.
- Identification of type of values that can be applied to particular operator and expression will be identified.
- Identification of the code block respective to which the expression will be identified.

Here figure 2 is showing the framework to analyze the program code and the specification under criteria as described. This analysis is shown in number of associated work stages. These work stages are given here under

A) Program Flow Analysis

The first stage of work is to analyze the program flow. The flow analysis is here defined in the form of control flow graph. Here the conditional statements and loops will be used to generate the branches over the program code and the input output statement as the straight statement as pre and post statement to the system. During this stage, the anlaysis to the program code is done to identify the valid statements and remove the non required program code. So that the filtered statements will be processed. This kind of program flow analysis comes under component extraction. The chacteristics of this kind of analysis includes:

- Separate the program types
- Generate the program flow
- Identify the work flow stages and test path
- Identification of unreachable statements
- Identificaiton of loops
- Filtration of repeated code

B) Expression Identification

The second stage of work is to identify the expression over the code. These expression identification includes the identification of:

- Variables
- Operators
- Operator types
- Data types

This kind of analysis comes under the component extraction and component search. The mapping of the functional code to the statements is also done in this section.

C) Operator Mapping

Once the expression are identified, the next work is to map these expressions respective to the operators and identify the operators and operators class that can be mapped to the code. This mapping identify the operator type such as procedural operators or object oriented operators:

- Type of data applicable to operator
- Operator input and output identification

- Restriction of operators
- Rules and acceptivity to the operator

D) Mutatnt Generation:

The final stage of work is to generate the mutation respective to the code. This mutatnt generation is done to analyze the code or code block under different values and aspects. These values and operators can be generated by generating the separate code segment and performing the data on it. The steps for mutation generation are given here under:

- Generate the operator
- Generate the operator specific code segment
- Identify the data applicability
- Generate the data cases
- Run the data
- Identify the mutant validity.

Based on this mutation maping the identification of the life of mutant and the operator code is done. This analysis is identified in terms of number of alive and dead mutants in the code.

IV. CONCLUSION

In this paper, the mutation process architecture is defined. This process is defined in the form of model and the work stages applied to analyze the code under mutation testing. The paper has explored the requirements and constraints on each stage.

REFERENCES

[1] John A. Clark," Semantic Mutation Testing", Third International Conference on Software Testing, Verification, and Validation Workshops 978-0-7695-4050-4/10© 2010 IEEE

[2] Mike Papadakis," An Empirical Evaluation of thes First and Second Order Mutation Testing Strategies", Third International Conference on Software Testing, Verification, and Validation Workshops 978-0-7695-4050-4/10© 2010 IEEE

[3] Alper Sen," Coverage Metrics for Verification of Concurrent SystemC Designs Using Mutation Testing", 978-1-4244-7806-4/10©2010 IEEE

[4] Fevzi Belli," A Formal Framework for Mutation Testing", 10 Fourth IEEE International Conference on Secure Software Integration and Reliability Improvement 978-0-7695-4086-3/10© 2010 IEEE

[5] K. K. Mishra," An Approach for Mutation Testing Using Elitist Genetic Algorithm", 978-1-4244-5540-9/10©2010 IEEE

[6] Lingming Zhang," Test Generation via Dynamic Symbolic Execution for Mutation Testing".

[7] Mohsen Fallah Rad," Implementation of Common Genetic and Bacteriological Algorithms in Optimizing Testing Data in Mutation Testing", 978-1-4244-5392-4/10©2010 IEEE

[8] Marinos Kintis," Evaluating Mutation Testing Alternatives: A Collateral Experiment", 2010 Asia Pacific Software Engineering Conference 1530-1362/10© 2010 IEEE