Score-Based Test Case Prioritization and Mutation Adequacy Check

J Santhosh
Assistant Professor, Department of Computer Science, Sree Narayana Guru College, K G Chavadi, Coimbatore, Tamil Nadu, India

Mini N
M.Phil Research Scholar, Department of Computer Science, Sree Narayana Guru College, K G Chavadi, Coimbatore, Tamil Nadu, India

Abstract – Software testing is an inevitable part of a Software Development Life Cycle (SDLC). Under the umbrella activity of software quality assurance, software testing plays a vital role in shaping the quality of software. It is aimed at providing high quality software (with minimal errors) that is cost effective. Often the cost of a product depends upon the time and efforts spend over molding it. In SDLC, since majority of the time and effort are utilized in testing the software, this phase incurs the major cost while producing quality software. Hence the focus remains on how to carry out effective testing with minimum cost. Over time, various software testing techniques have been devised and implemented. Each of them had its own pros and cons. Mutation testing is a white box testing technique that deviates from the usual testing approaches. When the usual white box testing techniques relies on different coverage areas (For example: statement, branch, path) based on control-flow or data-flow criteria, mutation testing behaves differently altogether, creating mutant programs for testing. Here, the goal would be to create such test cases that can distinguish the original program from its mutants. For effective testing, the test suite needs to be sufficient enough; clearly eliminating the idea of having large number of test cases that are similar in nature. Depending on the mutation score, the focus is on checking the adequacy of the test suite. It also aims to get the test cases prioritized, thereby improving the effectiveness of mutation testing.

Index Terms – Software Testing, Mutation Testing, Test Adequacy, Test case.

1. INTRODUCTION

In recent years software testing technologies have played an important role in the development of any application [1]. So it has emerged as a dominant software engineering practice which helps in effective cost control, quality improvements, time and risk reduction[2] etc. The growth of testing practices has required effective software testers to find new ways for estimating their projects efficiency. The key research area in this field has been measurement of the metrics for the mutation testing. Since mutation testing [3] plays a critical role in effective and efficient software development through the help of mutants, assessing the progressing the software development and testing process is very complex.

In this paper an improved, fast and innovative technique for Mutation testing tool has been developed. The proposed system measures the test suite adequacy criteria to detect the mutants effectively and quickly, together with the option of prioritizing the test cases. A variety of objective has been involved in the proposed system. One such function involves the software identifying the defects, that is, the failure of certain test cases at the time of execution of the mutant program. The proposed system describes a test case evaluation and prioritization technique [4] through which the mutants thus detected will be killed. The techniques used here report the empirical results measuring the effectiveness of this test suite, which can be used for comparison. This helps in reducing repeated testing with similar test cases in the mutation testing, which may have consumed much time and effort. Here the proposal uses a test case adequacy check and test case prioritization. They were compared with respect to their effectiveness for mutation results.

Test case prioritization helps in ordering the test cases for execution in a descending order of priority. In this way, the test cases with the higher priority, based on some adequacy criterion, are executed first, followed by lower priority test cases to detect mutants in mutation testing. From the existing test adequacy test and test case prioritization criteria code-based and model-based are considered for this research work. In Code-Based Test Case Prioritization [5], priority to test cases is assigned based on the source code in the system. Most of the test case prioritization methods are code-based. In Model-Based Test Case Prioritization and Mutation Adequacy Check (MBTCP_MAC), a system model is used to prioritize the test cases. The MBTCP_MAC may improve the early mutant detection as compared to the existing system. MBTCP_MAC may be an inexpensive alternative to the existing test case prioritization methods. In spite of its potential association between the range of mutants and the real fault detection capability, the mutation adequate test suite does not fully exploit the diversity.

2. PROBLEM DEFINITION

Some software defects [6][7] leads to failure only when certain local or non-local program variable interactions occur. Mutant program [8] can also make the output different and it should be treated well. So mutation testing is performed by changing some code in the program and testing with the same test suite. Mutation testing is important in testing because they reflect the differences in the test results. So, effective test case generation and performing test case adequacy criteria analysis on mutation testing is important. Apart from this, there is a huge need to analyze the quality of the test cases. There are several tools and techniques have been used in the literature related to the mutant analysis. In the paper [9] a detailed survey is described about the mutational testing and test case adequacy test analysis. Existing solutions for the mutation testing, and test case adequacy criteria verification systems are carried out as theoretical analysis, which only depends on the manually written test cases where the tester should enter the expected and actual results. So the system needs to convert the code into machine language to get the adequacy test on the test cases and have to find the equivalent mutant program effectively.

3. PROPOSED SYSTEM

Test case prioritization helps in ordering the test cases for execution in a descending order of priority. In this way, the test cases with the higher priority, based on some adequacy criterion, are executed first, followed by lower priority test cases to detect mutants in mutation testing. From the existing test adequacy check and test case prioritization criteria, Code-based and Score-Based are considered for this research work. In Code-Based Test Case Prioritization, priority to test cases is assigned based on the source code in the system. Most of the test case prioritization methods are code-based. In Score-Based Test Case Prioritization and Mutation Adequacy Check (SBTCP_MAC), a system model is used to prioritize the test cases. The SBTCP_MAC may improve the early mutant detection as compared to the existing system. SBTCP_MAC may be an inexpensive alternative to the existing test case prioritization methods. However, the existing SBTCP_MAC techniques do not consider the implicit dependencies arising due to object-relations.

To overcome this limitation an Extended Finite State Machine (EFSM) Score-Based Mutation Test Suite Minimization method using dynamic dependence analysis with test case mutation adequacy criteria score calculation is proposed. The proposed method automatically identifies the difference between the original model and the modified model as a set of elementary modifications or changes. This proposed method reduces the size of a given Mutation Test Suite (MTS) by examining the various interaction patterns covered by each test case in the given MTS. Whenever the software system undergoes modification, there is a need for mutation testing and while performing mutation testing many test cases appear to be redundant.

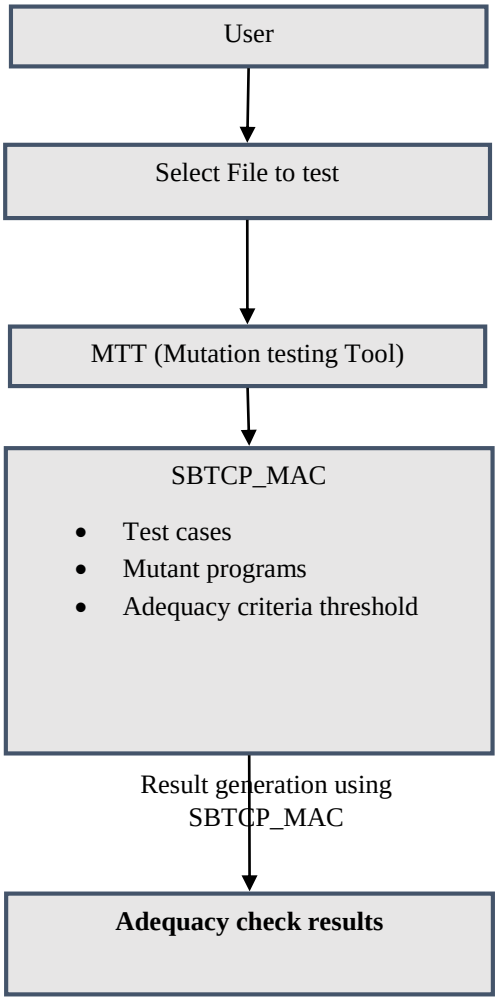

Figure 1.0 Proposed System Architecture

The research approach of Test Suite Minimization using Dynamic Interaction Patterns (DIPs) identifies redundant test cases and removes them and also the adequacy criteria is checked. Also, the research work attempts to improve the mutant detection ability by applying the dynamic dependencies in the place of static dependencies. The Score Based Testing (SBT) gives better results in test case prioritization, so the proposed system enhances the existing score calculation process with the rule prioritization. The reason is that ituses systematic approach Domain Specific Language (DSL) that supports higher level abstractions than general purpose modeling languages. Hence in this research work, an extended study on SBT known as domain specific SBT (DSMBT), is attempted. Experiments were conducted for the sample

programs developed in visual stuio.net framework. The results were compared with existing criterion check method.

3.1 Score-Based Test Case Prioritization and Mutation Adequacy Check (SBTCP_MAC)

The proposed system generates a new mutation testing tool with the score calculation and rule priority calculation functions and named as SBTCP_MAC. This allows the tool to select maximum priority test cases with the consideration of adequacy check. This will be performed by the fault detection. Test Suite Prioritization (TSP) facilitates development of complex systems by increasing the early fault detection capability and reducing the overall testing time to calculate the score and detect the adequacy. Reduced test suite is used for the system model and information on its execution is used to prioritize the test cases. Executing test model is comparatively less expensive than testing the entire system. Also less overhead is involved in test case prioritization using Score Based Testing (SBT) technique.

In order to find out interaction patterns, dependency based analysis is used in the system model. There are three types of dependencies namely structural, behavioral and traceability. Structural dependency involves dependencies among parts of a system. It includes system content, data and control. Behavioral dependency includes abstractions provided by the use of public interfaces and event broadcast. Traceable dependency covers inter-relationships between different artifacts namely dependencies between requirements, design and code. In structural category, data and control dependencies among the variables in the system are used to find Dynamic Interaction Pattern (DIP) whichis used to assign priorities associated with test cases in the system. Testing activities supported by dependency analysis consists of SBT, scenario-based testing and test suite reduction. Among the above three existing dependency analysis, SBT is taken for experimentation by using the technique called Dynamic Interaction Pattern (DIP) prioritization technique. Here after it is referred as Score-Based Test Prioritization (SBTP).

3.2 Dynamic Interaction Pattern (DIP) Prioritization Technique

System models are represented using Extended Finite State Machine (EFSM) which is an input for Score-Based Testing (SBT). From the model specification, Dynamic Dependency Graph (DDG) is formed based on the conditions which gives the data and control dependencies among variables (Bogdan Korel et al. 2002). By using DDG, the data and control dependencies called Interaction Patterns (IPs) are calculated which decides the priorities of test cases.

In order to prioritize test cases, the dependence analysis is used to identify different ways of the added/ deleted transitions interact with the remaining parts of the model. The principle of model dependence-based test case prioritization is to identify unique patterns of interactions between the model and the added/deleted transitions that are present during execution of the modified model on test cases. This information is used to guide the priority choice. During execution of the modified model on test, there are three possible types of interactions between a modified part of the model and the remaining parts of the model:

i. Effect of the model on modification termed as affecting transitions

ii. Effect of modification on the remaining part of the model called as affected transitions and

iii. Side effects of transitions caused by the modification.

These interactions may be viewed as computing a model slice. In the similar way there are three types of interaction patterns related to each modification (i.e., an added/deleted transition).

- An affecting interaction pattern
- An affected interaction pattern and
- A side-effect interaction pattern.

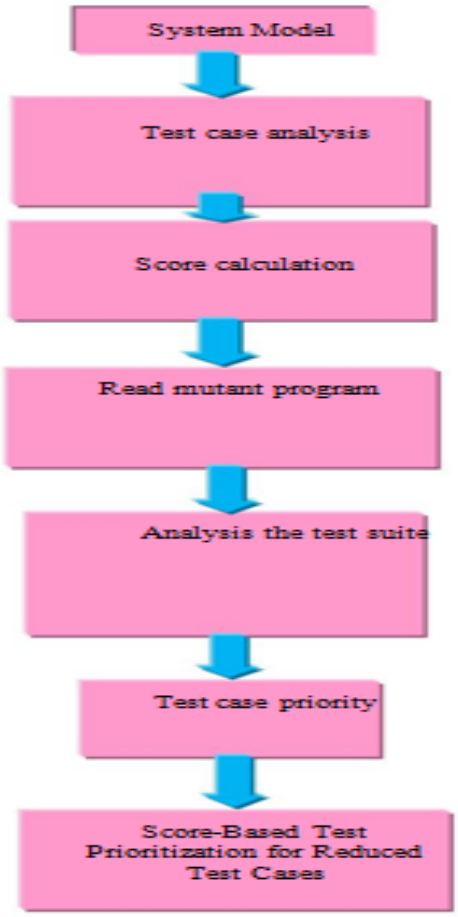

Figure 2.0 Score-Based Test Case Prioritization and Mutation Adequacy Check using DIP Technique

The affecting interaction pattern captures interactions between model transitions which affect the modification. The affected interaction pattern captures transitions which are affected by the modification. Finally, the side-effect interaction pattern captures interactions that occur because of side effects introduced by the modification. In this context, consider a side-effect to be an introduction of a new dependence or a removal of an existing dependence between other transitions. Interactions between model transitions are represented as model dependences between transitions. Consequently, the affecting interaction pattern, affected interaction pattern, and side-effect interaction pattern are represented as model dependence subgraphs (derived from a model dependence main graph) with respect to added and deleted transitions (Rachna and Arvind 2012). The system design using DIP technique is shown in Figure 2.0.

In existing prioritization technique, test cases are randomly selected and removed from the list whereas in proposed technique, based on priorities test cases are selected. The advantage of proposed technique is when interaction patterns are more, and then priorities will be high which will be assigned dynamically. Test cases with high priorities are executed first which detect more number of faults at the earliest.

Proposed Test case Priority calculation

Input: A set of interaction pattern test distribution

IPS={TS(IP1(t,T)),…,TS(IPq(t,T))}

A set of high priority tests: TSH

A set of low priority tests: TSL

Output: Prioritized test sequence, S

1. p=0
2. while true do
3. sort IPS in the descending order of number of T
4. for every test t in TS(IPi(t,T)) ∑ IPS do
5. if TS(IPi(t,T)) ≠ null then
6. p = p+1
7. remove test t from everyTS(IPi(t,T)) to which tbelongs
8. insert t into S at position p
9. if p =|TSH| then exit while loop
10. endif
11. if TS(IPi(t,T)) = null then
12. IPS = IPS – {TS(IPi(t,T))}
13. endfor
14. endwhile
15. for p=1 to |TSL| do
16. select randomly and remove test t from TSL
17. insert t into S at position p+|TSH|
18. endfor
19. output S

The above algorithm explains the process of priority calculation based on the score. Low priority test case execution is optional. If time permits, one or two low priority test cases are selected randomly and executed which will not have more impact on testing performance.

4. EXPERIMENTS AND RESULTS

The proposed system has been created a dataset of mutant and dataset1 executables for the Windows operating system. This mutant and dataset1 file collection was taken for the experiment. This acquired 20 mutant files from the Dataset1, including .cs, .vb files, were gathered from machines running the Windows operating system. The dataset1 set contained 100 file.. Some of the files in the collection were either compressed or packed. These files have added with the mutants, which have used the following operators.

Type of files for experiments	**Details**
Dataset(Normal Programs)	20 files
Generated Mutant files	20 mutant files
Number of operators used	+,-,/,* ,<,> etc.,
Total test cases used	25

Table 1.0 Dataset taken for experiments

The MTT applied on dataset1 software for data mutation detection purposes. These methods were proposed for automatic detection of mutant code and score by applying both the test cases based on the priority. Evaluation performed in these studies showed that rule prioritization and mutation detection with different metrics increase the detection accuracy. The proposed method can use such an approach in order to overcome quality issues of the software by detecting the mutants effectively. In addition, this would like to point out that classifying dataset1 files is also useful and can reduce the load of mutant files. Also, the large number of mutant programs in the dataset that could be dissembled indicates that in order to appear dataset1 and to pass metrics. The first set of experiments is to compare the performance of different test cases and calculates the score.

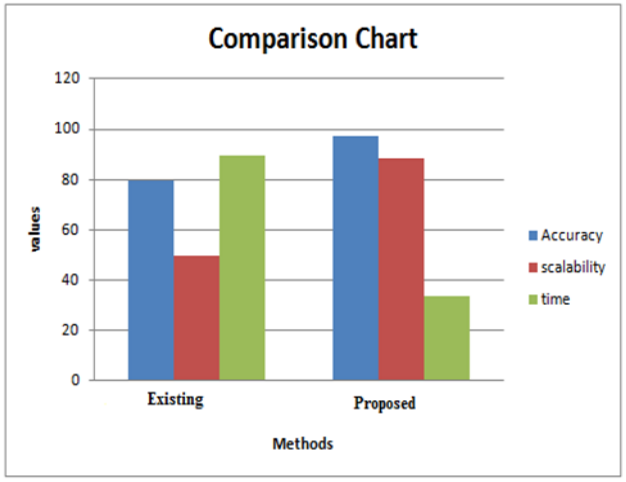

Figure 3.0 comparison chart

Figure 3.0 shows that all strategies perform significantly better than traditional mutant code detection technique and strategy with simple test suite priority and score calculation. Since the inclusion of additional techniques gives the better results in terms of accuracy, scalability and time. The accuracy is calculated using the expected result and the actual result by the tool. If the desired count is similar to the result, then the accuracy is high. From the theoretical aspects, the proposed system is more scalable and need less time to complete the test.

5. CONCLUSION

This proposed system introduces improved methods for score based mutation adequacy criterion and test case prioritization based on that. The proposed system calculates the adequacy criterion and a corresponding mutation score, called the distinguished mutation score based on its formal definition. The new system aims to reduce the time of testing and verifies the adequacy for every test case. This differentiates the mutation adequacy criterion and can be applied to detect the mutation in the program. This underlying relation provides theoretical evidence that the distinguished mutation adequacy criterion is more effective at detecting faults than is the traditional mutation adequacy criterion. We also provide an empirical evaluation of the mutation adequacy criteria in terms of their fault detection effectiveness, test suite sizes, and various score levels. This uses 20 programs. The mutation testing tool is implemented using C#.net. The test case priority has been experimented with the code based tests case methods and it gives optimal results. Comparison has been made between the existing system and the proposed system on the basis of test case adequacy and priority.

REFERENCES

[1] Lima, Igor Ribeiro, Tiago de Castro Freire, and Heitor Augustus Xavier Costa. "Adapting and Using Scrum in a Software Research and Development Laboratory." *Revista de Sistemas de Informação da FSMA,(9)* (2012): 16-23.

[2] Duvall, Paul M., Steve Matyas, and Andrew Glover. *Continuous integration: improving software quality and reducing risk*. Pearson Education, 2007.

[3] Jia, Yue, and Mark Harman. "An analysis and survey of the development of mutation testing." *IEEE transactions on software engineering* 37.5 (2011): 649-678.

[4] Elbaum, S., Rothermel, G., Kanduri, S., & Malishevsky, A. G. (2004). Selecting a cost-effective test case prioritization technique. *Software Quality Journal*, *12*(3), 185-210.

[5] Do, Hyunsook, and Gregg Rothermel. "A controlled experiment assessing test case prioritization techniques via mutation faults." *Software Maintenance, 2005. ICSM'05. Proceedings of the 21st IEEE International Conference on*. IEEE, 2005.

[6] Fenton, Norman, Martin Neil, William Marsh, Peter Hearty, David Marquez, Paul Krause, and Rajat Mishra. "Predicting software defects in varying development lifecycles using Bayesian nets." *Information and Software Technology* 49, no. 1 (2007): 32-43.

[7] D'Ambros, Marco, Alberto Bacchelli, and Michele Lanza. "On the impact of design flaws on software defects." *Quality Software (QSIC), 2010 10th International Conference on*. IEEE, 2010.

[8] DeMilli, R. A., and A. Jefferson Offutt. "Constraint-based automatic test data generation." *IEEE Transactions on Software Engineering* 17.9 (1991): 900-910.

[9] Jia, Yue, and Mark Harman. "An analysis and survey of the development of mutation testing." *IEEE transactions on software engineering* 37.5 (2011): 649-678.