Evaluate model based mutation technique to detect test cases error: A review

Jyoti Bhogwan
jyotibhogwan1@gmail.com
Parul Institute of Engineering and Technology, Vadodara, Waghodia, Gujarat

Tejal Patel
tejal.patel@paruluniversity.ac.in
Parul Institute of Engineering and Technology, Vadodara, Waghodia, Gujarat

ABSTRACT

In this paper, we study how model oriented programming and Mutation testing used to analyze the correct behavior of the system under the faulty data. In the previous time, many techniques have been proposed which perform testing automatically under the presence of faulty data in order to analyze robustness and efficiency of the system. The faulty test cases may affect the system testing which analyzes system robustness and efficiency. In this work, the faults from the generated test cases will be detected so that automated testing can be performing in the more efficient manner and detect a maximum number of faults in the system. Then, we recommend a set of mutation operators that are specifically propelled starting with these operations. By combining above two techniques, fault detection in test case data can be done efficiently.

Keywords: Software testing, Testing efficiency, Automatic test case generation, Mutation testing, Model-based testing

1. INTRODUCTION

The software is a non-tangible device like documentation and computer programs and it is different from the tangible hardware device. Software program Engineering is the field of laptop technology, which applies engineering principles to create, function, alter and hold software program components. Methodologies used for this work mainly are Mutation testing and Model-based testing, and by combining two techniques fault detection in test case data can be done efficiently.

Model-based is a utility of version-based design for designing and optionally also executing artifacts to carry out software program testing or device trying out. The model may be used to represent the desired conduct of a system beneath check (SUT), or to symbolize testing strategies and a take a look at surrounding [10]. Mutation testing is a sort of checking out in which, the utility is tested for the code that changed into modified after fixing a specific worm/illness [9].

Fig 1. gives an overview of our approach, which we refer to as model-based mutation testing [7]. Yellow parts highlight the aspects of mutation testing that we integrate into model-based testing, which is depicted in grey [7]. Model-based testing (MBT) is a black-box testing technique requiring no knowledge about the source code of the system under test (SUT) [7].
Only the interface to the SUT has to be known. A test engineer creates a formal model that describes the expected behavior of the SUT (Step 1) [7]. Test cases are then automatically derived from this test model. A crucial matter in MBT is the choice of the test criterion [7]. It specifies which test cases shall be generated and hence, has a great influence on the quality of the resulting test suite [7]. Exhaustive testing, i.e., using all of the test cases that can possibly be created from the test model, is impractical [7].

- **How will Mutation Testing win this?**

First, you produce mutants - variations of the program. A mutant is nothing, however, a program that is written as a deviation. It contains a self-sown fault.

Examples are:

- Logical connector replacement
- Relational operator replacement
- Statement removal
- Arithmetic operator replacement
- Absolute value insertion, etc.

![Figure-1 Overview of model-based mutation testing. [7]](image1)

<table>
<thead>
<tr>
<th>Test set data</th>
<th>Expected result</th>
<th>Mutant 1</th>
<th>Mutant 2</th>
<th>Mutant 3</th>
<th>Mutant 4</th>
<th>Mutant 5</th>
<th>Mutant 6</th>
<th>Mutant 7</th>
<th>Mutant 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>14 GP not assigned</td>
<td>Success-not assigned</td>
<td>Fail-GP assigned</td>
<td>Fail-GP assigned</td>
<td>Fail-GP assigned</td>
<td>Nothing happens</td>
<td>Success-not assigned</td>
<td>Syntax error</td>
<td>Success-GP not assigned</td>
<td></td>
</tr>
<tr>
<td>15 GP is assigned</td>
<td>Fail-GP not assigned</td>
<td>Fail-GP not assigned</td>
<td>Fail-GP not assigned</td>
<td>Fail-GP not assigned</td>
<td>Fail-GP not assigned</td>
<td>Fail-GP not assigned</td>
<td>Fail-GP not assigned</td>
<td>Fail-GP not assigned</td>
<td></td>
</tr>
<tr>
<td>13 GP not assigned</td>
<td>Failure-GP not assigned</td>
<td>Failure-GP not assigned</td>
<td>Failure-GP not assigned</td>
<td>Failure-GP not assigned</td>
<td>Failure-GP not assigned</td>
<td>Failure-GP not assigned</td>
<td>Failure-GP not assigned</td>
<td>Failure-GP not assigned</td>
<td></td>
</tr>
<tr>
<td>13 GP not assigned</td>
<td>Failure-GP not assigned</td>
<td>Failure-GP not assigned</td>
<td>Failure-GP not assigned</td>
<td>Failure-GP not assigned</td>
<td>Failure-GP not assigned</td>
<td>Failure-GP not assigned</td>
<td>Failure-GP not assigned</td>
<td>Failure-GP not assigned</td>
<td></td>
</tr>
</tbody>
</table>

![Figure-2 mutation testing [8]](image2)

2. RELATED WORK DONE TILL NOW

An investigation is required of how to apply mutation analysis on a complex software system, as well as demonstrating the value of doing so, even on well-tested systems. The experiences are presented using mutation analysis on the Linux kernel’s RCU (Read Copy Update) module, where the existing techniques can be adapted to constrain the complexity and computation requirements [1]. It is seen that mutation analysis can be a useful tool, uncovering gaps in even well tested modules like RCU. Using different mutation operators or tools could lead to different results. In future work, there is a need to make enhancement such that the type of mutation operators or tools being utilized in the method do not affect the results.

There are numbers of possible configurations due to which reason the generating and testing of all mutants for each program is not feasible. The possibilities of reducing computational costs for mutation testing in configurable systems are discussed and the static analysis and T-wise testing are utilized. The proposed approach supports efficient mutation testing for configurable systems. There is lack of an open-source program with test cases and an oracle to be used in such evaluations [2].
A new two-way crossover and adaptable mutation method are proposed that intelligently use the fitness information to generate fitter offspring. Experiment results prove that our proposed approach can find the optimal test cases in less number of attempts (reduces computational cost) [3]. Besides that, it can detect software bugs from suspiciously equivalent mutants and these mutants eventually get killed (increases mutation score) [3]. With these proposed changes, the object gains the desired state quickly and methods receive required arguments [3]. Here, only the eMuJava v.2 tool is used which does not make sure that the method will support multiple evolutionary approaches [3].

A recent benchmark, called CoREBench1 [4] is used to investigate the relation between four popular test criteria (strong mutation, weak mutation, statement coverage, branch coverage) with fault revelation. The useful mutants are detected with the help of designed technique and the whole mutation testing process is automated. The results show that mutation test suite finds considerably more faults than other existing criteria. The mutation testing and symbolic execution are performed within the real-world software systems by ensuring the scalability of the system. The dynamic symbolic execution is applied to efficiently and automatically generate mutant-adequate test suite and detect subsuming mutants for mutation testing.

The standard testing techniques are often perceived as boring and difficult when compared to creative programming and design activities which dominate education. The use of the CODE DEFENDERS game concept is proposed to develop systematic exercises that teach a range of different testing concepts. It is seen that the learners not only acquire better testing skills but will in the long term become better software engineers. As players progress through levels of this game, they incrementally learn and practice testing concepts [5]. A mapping of core developer testing concepts, such as statement or branch coverage, to categories of puzzles in the framework of the game [5]. The approach might be of increasing difficulty which might lead to confusion (too challenging) or frustration. The challenge ahead is to produce suitable games that exercise the testing concepts that already exist.

Mutation testing is one kind of the software testing in which some mutants are injected intently in the program/software and tester finds these mutants during the testing process. A Hybrid Genetic Algorithm is proposed for generating test data automatically using data flow testing approach for mutation testing. The proposed method produces maximum mutation score. On comparing the random and genetic algorithm, it is found that mutation is better with the hybrid genetic algorithm. The mutation score achieved here is maximum and thus helps in proposing an efficient improvement within the existing work. Only a few genetic algorithms are tested here and the experiments do not ensure that there can be improvements achieved with the application of any other algorithm. The study is extended further, by applying the other optimization methods.

There is a need to show how to implement the model-based mutation testing technique more efficiently to timed automata. The tool Eedar is utilized here which belongs to the well-known UPPAAL tool family. The quality of the test suite is maintained with respect to the mutation score and its quality is improved with respect to adaptiveness. The proposed approach is compared to an existing implementation of model-based mutation testing for timed automata, which showed a high-speed up. The quality of the test suite is maintained with respect to the mutation score and its quality is improved with respect to adaptiveness. There is a limited feature provided in the Eedar method, which limits its usage. In future work there is a need provide further features to Eedar, including an automated way for turning models input enabled, providing more information on the produced strategies and a model-mutator.

Popular mutation testing tools use a restrictive set of mutants that don't adjust to the community standards and mutation testing literature. Associate in a Nursing extended set of mutants is made and enforced by a well-liked mutation-testing tool named PIT for finding the arising issues. The results confirmed that the extended mutant’s area unit simpler than the initial version of the PIT and 2 alternative common mutation testing tools. The results demonstrate that the extended mutant’s area unit simpler by twenty-third, twelve-tone system and seven than the mutants of the initial PIT, major and muJava. The extended mutant’s area unit a minimum of as robust because of the mutants of all the opposite 3 tools along [9]. In this research, the PIT version is not publically available and thus the results achieved only determine the efficiency of proposed work in limited areas. To support future research, the new version of the PIT is made which is equipped with the extended mutants, publicly available.

There is a desire to make a powerful check suite for testing an exact non-functional property with the assistance of acceptable mutants. The construct of targeted mutation is projected that focuses mutation effort to those components of the code wherever an amendment will create a distinction with relation to the targeted non-functional property. it’s seen however targeted mutation is applied to derive economical check suites for estimating the Worst-Case Execution Time (WCET). The program slicing to direct the mutations to the components of the code that square measure seemingly to own the strongest influence on execution time. The proposed approach in a more general context than just execution time. Although the proposed work has provided efficient results in identifying mutants that have non-functional properties but the method used limits the applications. An enhancement can be made in future related to the generation of test-suites with more accurate concepts involved.

Although several efforts are created to scale back the machine price of mutation testing, its measurability issue remains in to observe. Formal conception Analysis (FCA) to cluster the mutants along that have the best similarity in state infection against all look at cases. This technique selects one mutant from every FCA grouping and so selects look at cases to hide the mutants. The initial results show that the planned technique will cut back the execution time by eighty-three.93percentage with solely zero.257% the loss in preciseness. This approach provides a reduction in the high computational cost of mutation testing. The proposed approach is limited to only few test suites and uses only one compression method, which does not ensure its effectiveness in other scenarios as well. In future, this method can be extended so that the approach can be generalized to other test suites and other data compression methods can be investigated.

It is important to identify the Equivalent mutants, which are live mutants that do not exhibit a different output from the original program’s output, no matter what test input is given. A Fuzzy model is proposed for weak and strong mutation testing to find out whether a mutant is equivalent or not. Both weak and strong mutation testing models have a common output where it determines the probability of a mutant being equivalent. Three parameters are taken for weak mutation testing which is Reachability, Infection, and Propagation as the input and three for strong mutation testing which is Reachability, Infection, Propagation and Impact. The probability of a mutant being equivalent is determined in case of two different mutation-testing models. The tests are performed by only including three limited parameters in the experiments. The future work of this paper is that more input parameters can be proposed for both the models to obtain results that are more accurate.
Every check criterion needs the generation of check cases that turns to be a manual and troublesome task among the mutation testing method. An automatic check generation approach is generated victimization hill ascension, for a sturdy mutation that aims at powerfully killing mutants, by that specialize in mutants’ propagation. The projected approach produces effective check information able to powerfully kill the bulk of mutants on C programs in an exceedingly bit of your time. The projected approach achieved the next sturdy mutation score than random testing, by 19.02% on the average, and also the antecedently projected check generation techniques that ignore mutants’ propagation, by 7.2% on the average. The proposed approach increases the mutation score and this in only a small number of iterations [6]. The search-based technique requires an appropriate fitness function to guide the search for different points in the search space. There can be an improvement of the fitness functions, ID part, in particular, to generate test data towards exposing a mutant in an observable output.

3. OBJECTIVES

The main objective of this research is described below:

- To study and analyze various mutation operators for generation of faulty data in software testing.
- To design and implement a technique for detection of faults from the test cases, which are generated using model, based mutation analysis.
- Implement model based mutation analysis technique for generation of faulty test data and compare with the proposed technique in terms of accuracy, execution time.

4. CONCLUSION

We combine model-based testing with mutation testing to generate a test suite that guarantees the coverage of certain modeled faults. The model-based mutation analysis is the technique, which generates complex and faulty test data to perform automated testing. In the presence of faulty data, it is very difficult to perform efficient testing and analyze system robustness. In this work, the generated test cases will be revealed subtle errors that have been found neither by manual testing nor by the operation of the system over several years.

5.REFERENCES

[7] An overview of our approach model-based mutation testing
https://pdfs.semanticscholar.org/9333/2db9a23b8beb7d3426dad229d2fb790b59.pdf, accessed 15th August 2017