Improving State Invariant Test Oracle Strategy using Mutation Analysis

M. Venkata Hari Prakash[#1], Dr.A.Ananda Rao [#2], Dr. P. Radhika Raju [#3]
[1]*M.Tech Scholar, Department of CSE, JNTUACEA, Ananthapuramu, A.P, India*
[2]*Professor, Department of CSE, JNTUACEA, Ananthapuramu, A.P, India.*
[3]*Ad-hoc Assistant Professor, Department of CSE, JNTUACEA, Ananthapuramu, A.P, India.*

Abstract:
A model should be designed with some specification languages. Such specification languages are transformed from the customer requirements to testable requirements. In model-based testing, abstract tests are generated from the UML model. These abstracts tests are transformed into the concrete tests with test inputs and to check the outputs with test oracles. This research mainly focused on possibility of the failures and exhibits the failures. In this research, the state invariants of the model are identified. State invariants are detected from the state machine diagram. The test oracles are measured in two parameters: precision and frequency. The mutants are generated from the test data. The mutant's reports are generated from the test case. In agile process, the test oracles are always not available because requirements are frequently changed. This research specifies generation of few adequate mappings, because instead of generating more inadequate mappings.The test cases are utilized to kill the generated mutants whenever the behaviour of the system changes.

Keywords: *Coverage Criterion, Model-based Testing, State Invariants, Test Oracles, Test Oracle Strategy*.

I. INTRODUCTION

Software testing is a part of SDLC (Software Development Life Cycle). Initially, Software testing is focused on finding the faults by running more tests on an application, or a system. Software tests are generated by using two artifacts, they are test inputs and test oracles. Test inputs states that the test values for method calls of an application. Whereas the test oracles specifies the analysis of the outputs i.e., whether the tests are satisfied or not.

The tests are generated by satisfying the different coverage criterion. Whenever the tests are running in an application, if a fault may be raised then the fault caused to raise an error then the fault is treated as a failure. The failure test or part should reveal to the tester, otherwise it leads to an effortless work [2]. The failure detected for an entire process is known as RIPR (Reachability, Infection, Propagation and Revealability) model or FF (Fault and Failure) model. The main focus is on coverage criterion, and this can be compared with different criterions. In unit testing, a small module of the software should be tested and the module is error free then it was integrated into the real-time software or desired working software. The module which had been tested effectively, then the module will be delivered to the end user.

Up to now the process is in normal software development process, when the software is developed in Agile or Development Operations (DevOps) there will be a continuous changing and adding the requirements. DevOps provides the collaboration between development phase (plan, code, build, test) and operational phase (deploy, operate, monitor).

The devops mainly focuses on automation; some of the tools are used in software testing. They are JUnit [3] and selenium [4].In this research, the Finite State Machine (FSM) model is considered. A model consist a set of states and transitions. The model was designed with some specification languages. The specification languages are transformed the customer requirements into mathematical form. The specification languages like Object constraint Language (OCL), Z Notation, B and VDM (Vienna Development Method)[4, 5, 6] are essential.

The Test Oracle problems are classified into four categories: they are specified test oracles are designed from specification requirements, derived oracles are designed from other sources like documentation or notes, implicit oracles get the information from different sources without domain knowledge, no test oracles or human oracles means there is no source to get the information [7].

This paper navigates the fundamentals of software artifacts in oracle problem in section 2, related work in section 3, test data generation in section 4, mutation analysis and mutant generation in section 5, experimental studies in section 6 and conclusion and future work in section 7.

II. FUNDAMENTALS OF SOFTWARE ARTIFACTS IN ORACLE PROBLEM

In this section, some of the basic terms defined in section 2.1, test oracle and test oracle strategy in section 2.2, subsumption of coverage criterion 2.3 and FSM and state machines in 2.4.

A. Basic Definitions

Def 1: **Model**: Model represents partial behavior of the software. It is designed using some specification languages.

Def 2: **State Invariants**: Invariants means true condition, which represents possible value to the state.

Def 3: **Test Requirements** (TR): Software artifacts are purifies into the test requirements. Those test requirements are used for creating test paths.

Def 4: **Test Inputs**: Test inputs are test values for sequence of method calls on program under test.

Def 5: **Expected Results**: these are analyzed results for the program logic.

Def 6:**Coverage Criteria**: It means providing some rules to create test requirements. Graph coverage criterion is applied for design structures, control flow graphs, use cases and FSM's and state machines.

Graph Coverage consist nodes and edges. In our context, Graph coverage consist states and transitions because in this research FSM is used for generating test paths. Each graph has an entry state and an exit state. The graphs are classified into two types: They are deterministic and non- deterministic graphs.

Def 7: **Mutation analysis**: Mutation analysis means small changes in the system behavior or program functionality. Mutation testing is applied on method based and class based attributes.

Def 8**Mutation score**: The ratio of mutants is killed from total mutants.

B. Test oracle and Test Oracle Strategy

Def 9: **Test Oracle**: test oracle is looks like a program. It compares the actual results and expected results, when both results are true then the test case is passed. Otherwise it is failed. Test oracle decides the test case is passed or not. Some of the steps to be followed to understand the test oracle are:

- Create test inputs.
- Run the program with test inputs.
- Verify the outputs are equal or not.

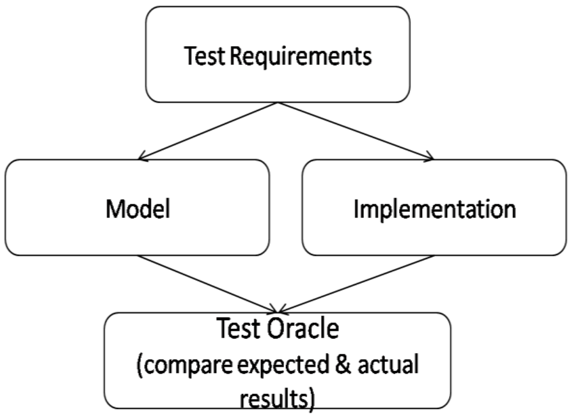


Figure 1: Test Oracle

Def 10: **Test oracle strategy**: In test oracle strategy, checking the test paths which states are revised and how many states are frequently checked is to be happened. It is measured with two parameters, one is precision and the other is frequency. Precision indicates which states to check in the program by using test oracle. Frequency indicates how many times the each state is to be executed. The frequency checks the states which are already checked in the precision for the purpose of revision.

UML is a modeling language in software development. State machine diagram is one of the UML behavioral diagrams. State machine diagram consist a set of elements like states, transitions, initial state, final state, state invariants (constraints or conditions). State invariant specifies boolean values (true or false). If the state invariant is true then the condition reaches as specified. Otherwise the condition is not satisfied. State invariants are identified in abstract tests. The abstract tests are generated from the UML state machine model.

Precise oracle is specified in the concrete test. The concrete tests are generated manually, in other words, some of the input values are required when tests are generated. These test inputs are often treated as a precise.

C. Subsumption of Coverage Criterion

Subsumption means one coverage criteria is covered by other coverage criteria. In other words, A and B are the two coverage criteria's. If A and B have some rules, but the A coverage covers B coverage i.e., if A subsumes B.

The structural coverage consist some coverage criterion. They are node coverage, edge coverage, edge-pair coverage and complete path coverage. The data-flow coverage is classified as all-defs coverage, all-uses coverage, all-du coverage, simple round trip coverage, complete round trip coverage, prime path coverage, complete path coverage. Coverage criterions are denoted as node coverage (NC), edge coverage (EC), edge-pair coverage (EPC), all-def's coverage(ADC), all-uses coverage (AUC), all-DU coverage (ADUC), Prime path coverage(PPC) and Complete path coverage(CPC).

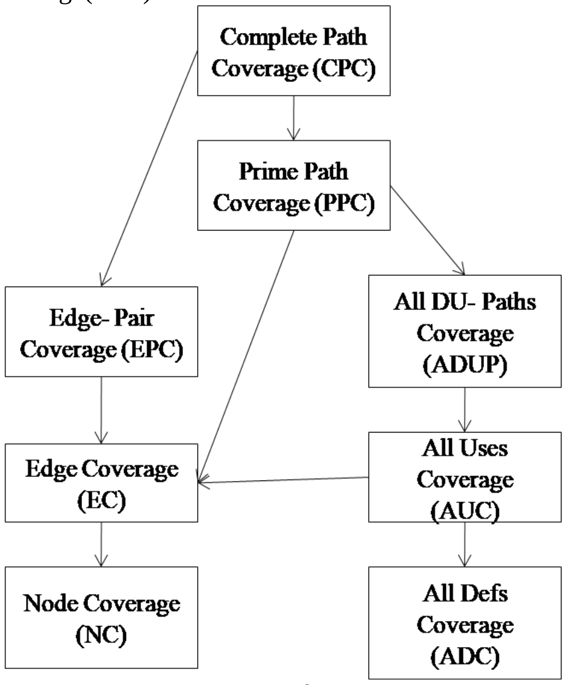

Complete Path Coverage (CPC)
Prime Path Coverage (PPC)
Edge- Pair Coverage (EPC)
All DU- Paths Coverage (ADUP)
Edge Coverage (EC)
All Uses Coverage (AUC)
Node Coverage (NC)
All Defs Coverage (ADC)

Figure 2: Subsumption of Coverage Criterion

The above figure is subsumption of graph coverage criterions. Subsumes coverage's are:

- EC subsumes NC.
- AUC, EPC, PPC subsume EC.
- EPC subsume CPC.
- AUC subsume ADC.
- ADUP subsume AUC.
- PPC subsume ADUP, EPC
- CPC subsumes PPC.

D. FSM and State Machines

System is designed in mathematical model, it is known as FSM (Finite State Machine). FSM model reduces the complexity by making the system into simplifying assumptions. Some of the assumptions are as follows:

- The system model designed with limited number of finite conditions is known as states.
- In the specified system, a state is to be performed what the state is specified.
- A state changes the conditions only in a number of finite ways it is treated as transitions.
- Events are the response to the system.
- Transitions execution time is less than zero(approximately).

A FSM is designed with a set of assumptions in mathematical model. The model consists only true conditions (possible conditions). The FSM is designed by using two ways mealy and Moore FSM machines. Mealy machine outputs depend on current state and current inputs. It has less state than Moore machine. It is ease to design mealy FSM machine. The Moore depends on current output state. It is somewhat complicate to design Moore FSM.

State machine consists true conditions and false conditions. A state is transformed to another state based on the two conditions. Sometimes state machines have two states to go to its possible state(true condition) and impossible state(false state). These conditions are represented by using constraints languages.

III. LITERATURE SURVEY ON TEST ORACLE PROBLEM

Kamaraj and arvind [8], is described different type of test oracles strategies. The test oracle strategies are specified on different domains like specification, documentation, heuristics, consistency, statistical and model based test oracle. Here, the heuristics ishaving more accuracy with 98%. The statistical and model based test oracles are having less accuracy with60%.

ShadiG.Alawneh and Dennis K. Peters [9], is created the test oracles using the specification based test oracles with JUnit. In this research, the test oracle is created using OMDoc (open mathematical documents).

IV. TEST DATA GENERATION

The test data generation means combination of test input generation and test oracle generation.

A. Test Input Generation:

The test inputs are given manually to the model. The test input data is generated automatically. The test inputs are nothing but test values. The test values are assigned to the model by using STALE (structured test automation language framework). The STALE is specially designed for FSM and the STALE imports the UML finite state machine diagram. The STALE exhibits all transitions in specified FSM. Each transition performs the specified operation.

The STALE reads the test inputs manually. The test data is generated automatically. In other words, the abstract tests are generated automatically and the concrete tests need some input values so it needs input values manually.

In the above represented figure 3 the process can be done to implement the research. This process states that, considering the Test requirements as a basic thing and framing those requirements in to the UML state machine diagram. From that diagram the Graph can be generated to accomplish its task, using the graph the Abstract tests will be gathered. Through those abstract tests the concrete tests are transformed, and mapping concept is applied as input parameters of it.

After this the concrete code is generated by following the before process. Finally the specified results, Test Oracles are outlined with the involvement of the JUnit test Assertions.

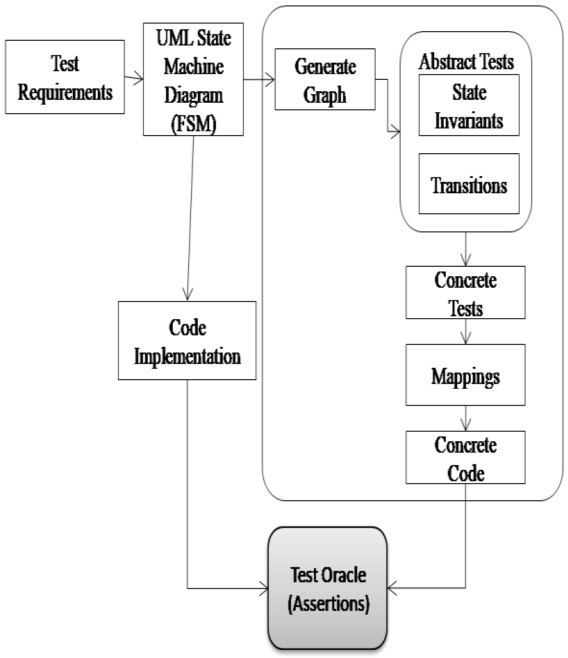


Figure 3: Test Oracle Generation in Model-Based Testing.

B. *State Invariant Test Oracle Generation:*

The state invariants are nothing but pre-conditions, post-conditions, constraints, predicates and conditions [14]. In test oracle problem is need to detect more invariants for state invariant oracle strategy. In state invariant test oracle strategy needs to identify the invariants from the model and write more test oracles i.e., writing the assertions to verify the state invariant.

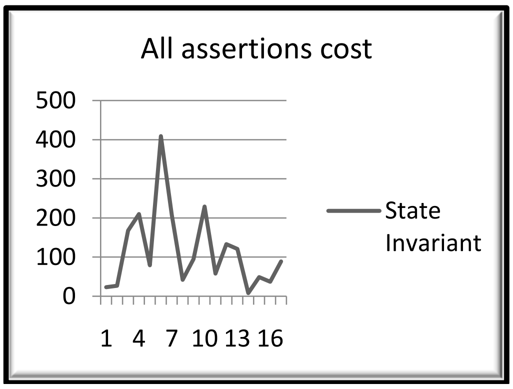


Figure 4: Number of Assertions to each Application

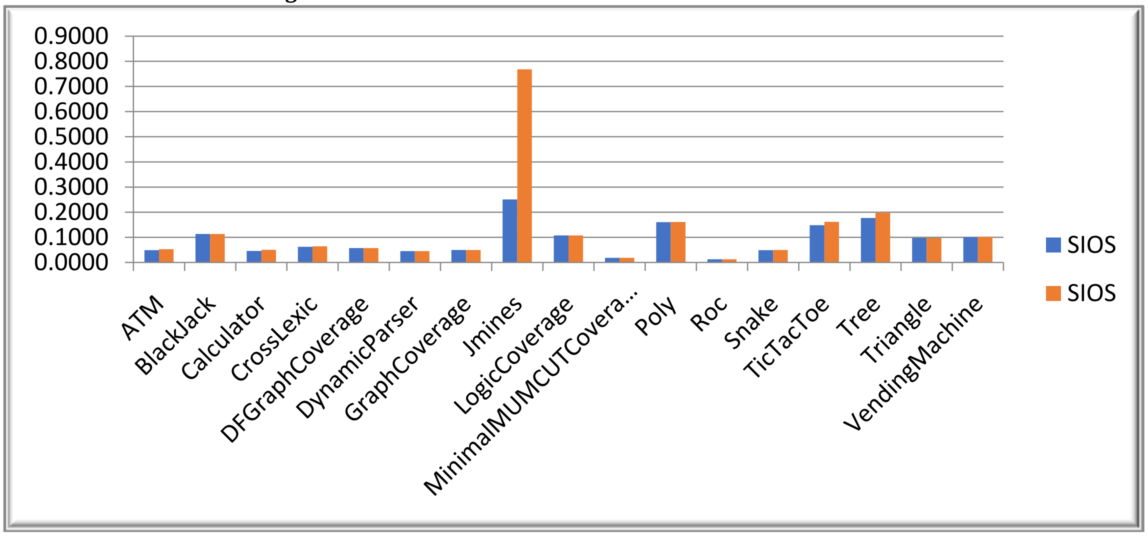


Figure 5: state invariants Comparing with Two Coverage Criterions

V. MUTATION ANALYSIS

Mutation analysis states that any kind of changes is applicable on a specified program then it performs actual behavior or not. When mutated program is tested with a test case, then the test case can be performed is-as original program. This process is called as equivalence mutants.

When the mutated program functionality raises the errors as specified by the generated mutants, those are called killed mutants. Mutation density of the program can be calculated as:

$$\text{Mutation density}=\frac{\text{Number of mutants proposed}}{\text{LOC}}$$

Where, LOC is Lines of code (according to their respective program) and Number of mutants specifies how many mutants are generated for a respective program.

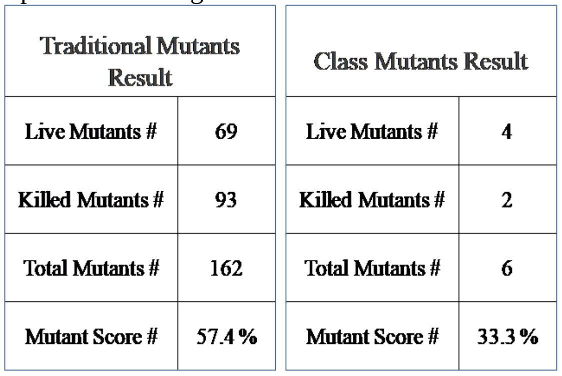
Mutation Cost is measured in:

$$\text{Mutants Score} = \frac{\text{Number of mutants killed}}{\text{Number of mutants proposed}}$$

The mutation cost is decided number of mutants killed over the total mutants. The mutants are killed by writing test oracles.

A. Mutants Generation and Reports

Mutants are generating by using mutation tools. Each language have their individual mutation tools. Some of the mutation tools for java is µJava, PITest, Jumble, Jester and Judy. Each mutation tool has their mutation operators. Mutation operators like arithmetic, logical, unary operators etc. the mutants report shown in figure 6.



Traditional Mutants Result		Class Mutants Result	
Live Mutants #	69	Live Mutants #	4
Killed Mutants #	93	Killed Mutants #	2
Total Mutants #	162	Total Mutants #	6
Mutant Score #	57.4 %	Mutant Score #	33.3 %

Figure 6: Mutants Report

VI. EXPERIMENTAL STUDIES

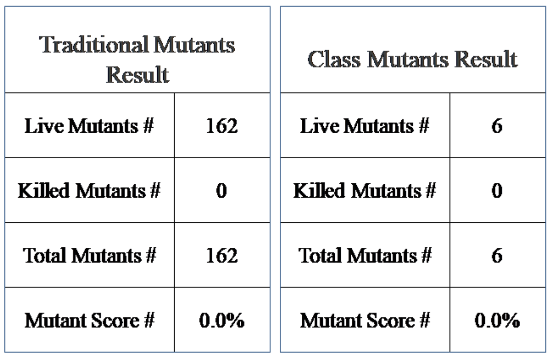
RQ1: which coverage criterion is effectively providing more mappings for a model?
RQ2:does the mutants creating is effective or not?

Table 1: Generation of Mappings

Coverage	**Vending Machine paths**
node coverage (NC),	6
edge coverage (EC),	9
edge-pair coverage (EPC)	16
all-def's coverage(ADC)	2
all-uses coverage (AUC)	4
all-DU coverage (ADUC)	6
prime path coverage(PPC)	63
Complete path coverage(CPC)	0

RQ1 satisfies the model elements. Each coverage criterion will provide some mappings for a model. The mappings are generated by using model transitions. Those model transitions are known as test paths. When the coverage criterion is providing more mappings then it is stronger coverage criterion. The coverage criterion gives less mappings then it is weaker coverage criterion. In table 1, three types of coverage's are listed. They are graph coverage, data flow coverage and logic coverage.

The mutant generation is good aspect in software testing. Always mutants generation is either effective nor not. When the more mutants are generated to the system then the more test cases are created and kill the mutants, this process is more effective than less mutants killing. When the more mutants are generated but it is not possible to killing the mutants. Which test case is killed more mutants then it is called effective test case. Otherwise, it is weaker test case. Example mutant report of weaker test case shown in figure 7.



Traditional Mutants Result		Class Mutants Result	
Live Mutants #	162	Live Mutants #	6
Killed Mutants #	0	Killed Mutants #	0
Total Mutants #	162	Total Mutants #	6
Mutant Score #	0.0%	Mutant Score #	0.0%

Figure 7: Weaker Test case of Mutant Report

VII. CONCLUSIONS AND FUTURE SCOPE

Software testing follows observability and compatibility. In model-based testing, a model needs effective state invariants. In this research, edge-pair coverage is the strongest coverage than edge coverage and prime path coverage. Always generating effective less more mapping are useless, but effective few mappings are generated by using complete path coverage.

In mutation analysis, Always generating more mutants is doesn't matter, but how many mutants are killed by using test case. If the test case is killed more mutants then it is best test case. The test case consists distinct assertions. Whenever killing less mutants then it consider as worst test case.

In future work, the STALE and µJava to be automated. Try to satisfy complete path coverage.

Provide techniques for identifying state invariants from the model.

REFERENCES

[1] Aditya P. Mathur, Foundations of Software Testing, India, 2013.

[2] JUnit, https://junit.org/junit5/

[3] Selenium, https://www.seleniumhq.org/

[4] Object Constraint Language (OCL), https://www.omg.org/spec/OCL/About-OCL/

[5] Z Notation, https://cse.buffalo.edu/LRG/CSE705 /Papers/Z-Ref-Manual.pdf

[6] Maria Grammer, A study in Formal Specifications, Spring 2015.

[7] http://www.greggay.com/courses/spring16csce747/Lectures/Spring16-Lecture11TestOracles.pdf

[8] KamarajKanagaraj and ArvindChakrapani ,Strategies of Automated Test Oracle – A Survey, AENSI publication, 2017.

[9] ShadiG.Alawneh and Dennis K. Peters, specification-based test oracles with junit, 2010.

[10] http://www.greggay.com/courses/spring16csce747/Lectures/Spring16-Lecture11TestOracles.pdf

[11] Nan Li and Jeff Offutt, Test Oracle Strategies for Model-based Testing, IEEE transaction of software engineering, 2016.

[12] Christian Burghard, Model-based Testing of Measurement Devices Using a Domain-specific Modelling Language, thesis report, Graz University of Technology, 2018.

[13] Thierry TitcheuChekam, Selecting Fault Revealing Mutants, University of Luxembourg, 2018.

[14] P.Radhika Raju, Dr. A.Ananda Rao, Optimization of program invariants, ACM SIGSOFT Software Engineering Notess, Vol.39, Issue 1, January 2014.