ISSN 2229-5518

AUTOMATIC TEST CASES GENERATION USING BEHAVIOUR ANALYSIS FOR SPECIFICATION MINING

Mahendran.N, Kamalraj.R, Karthik.S

Abstract- Software testing refers to the process of validating and verifying a software product to meet the requirements in design and development. Specification mining deals with the extraction of high level specifications from the existing code. To have mined specifications dynamic specification mining is used to infer common properties executions. To enrich the specification TAUTOKO tool is used to generate the test cases. This tool could not handle the methods and unable to generate call to methods and unable to modify the test and cannot observe the method call in the desired state. To deal this limitation integration of special test cases is done for method invocation that are invoked during pre-processing stage. Integration of test cases, which only deal the method calls and method specific operations for behavioural analysis of the executable program. It will check the methods and its return variables.

Index Terms: *Finite state automata, Specification mining, Type state miner, TAUTOKO*

1 INTRODUCTION

Software testing is done by two Types. They are manual testing and automated testing. Manual testing is the process of manually testing software for defects. It requires a tester to play the role of an end user and use most of all features of the application to ensure correct behaviour. Automated testing is the software tested to control the execution of tests.

Specification mining extracts the high level specification from the existing code. These high level specification are described by their behaviour and structure of the program. The test cases are generated by inferring the commonly observed behaviour of the programming structure. Observed behaviour are put into common class under test for generating the test cases. Various tools used for generating test cases are ADABU,DAIKON.

To infer the transitions between program structure and generate test cases TAUTOKO tool is used. TAUTOKO is a open source tool to generate test cases for specification mining. This Typestate miner generates the test cases by observing the behaviour of the programming model. The behaviour represents the branch coverage , return variables.

The commonly observed behaviour are put in common class under test and it is mutated to perform for generating the test cases. Main issues in this technique is that it cannot generate the test cases for method invocation and cannot synthesis parameter values. So it unable to produce more number of fault transitions during testing.

2 RELATED WORK

2.1 GENERATION OF TEST CASES: STATIC SPECIFICATION MINING

An approach for generating test cases for static specification mining is researched. The idea was to combine the specification mining with the test case generation. The core is to provide a generic feedback loop framework where specifications are fed into a test case generator, the generated tests are used to refine the specifications, and the refined specifications are again given as input to the test case generator. Extension of the work is done by providing an implementation of the framework for typestate mining, as well as an evaluation of how useful enriched specifications are for a real-world application. There is a large body of work on test case generation. Several approaches use simple randomized algorithms to generate tests. Ciupa et al apply random testing to several industrial sized applications.Symbolic execution simulates execution of the program using symbolic values rather than concrete ones and relies on constraint solvers to derive test data [10].

2.2 USING TEST SUITE TO GENEARTE TEST SUITE

It is done by mapping STRIPS planning language. Test cases are automatically generated from use cases by formal transformation of a detailed use case description including pre- and post conditions to a UML state model , Generation of test cases from the state model. The Preconditions and Postconditions sections of the use case template allow us to specify the contract of the use case. Preconditions describe verifiable conditions, which must hold before the execution of the use case. The preconditions of the use case as constraints on the first state representing the use case. Postcondition on a state can be modelled using a superstate with two substates.

These actions formalise the idea that the use case establishes the postconditions on successful completion. These action statements during test suite planning, when the actions establishing a condition with the preconditions

of other use cases or steps requiring this condition. The test sequences derived from the state machine are consistent in the sense that the preconditions of all transitions in the sequence are satisfied. A coherent procedure to derive test cases from use cases in a formal and partly automatic way by the expected system responses have to be added to the test sequence manually to yield complete test cases[5].

2.3 TEST CASES GENERATION BY MUTANTS

A new symbolic procedure for the automated generated of test cases from a set of mutants using bounded model checking . Mutation testing is a powerful testing technique based on the idea of making changes to a syntactic description of a computing task and deriving test cases from these changes. The changes mimic mistakes programmers or designers make during the description of the computing task. Mutation testing provides a fault-based test criterion, called mutation adequacy, i.e., a rule imposing test requirements on the test bench that good test cases should examine. Program-based mutation testing forms a three step test process as given a program and a test bench. The program is seeded with artificial faults according to a fixed fault model. Each seeded fault is kept in an individual copy of the original program source, called mutant. Each test case from the test bench is then executed on the original program and on its mutants. The problem of automatically generating test cases from undetected faults is typically not addressed by existing mutation testing systems. To overcome this a symbolic procedure SymBMC usedfor the generation of test cases from a given program using Bounded Model Checking (BMC) techniques. The SimplifiedBMC procedure attempts to generate a test case for a program and one of its mutants, whereas the SymBMC procedure generalizes the approach to generate test cases from a set of mutants[2].

2.4 MODEL DRIVEN APPLICATION FOR DERIVING TEST CASES.

The proposed approach is to generate test cases for graphical user interface applications. GUIs lend themselves to the feedback-based approach for producing test cases that exhaustively test only two-way interactions between GUI events. GUI testing is important because GUIs are used as front-ends to most software applications and constitute as much as half of software's code. A correct GUI is necessary for trouble-free execution of the application's underlying business logic. Finite state machines have been used to model GUI. A GUI's state is represented in terms of its windows and widgets and each user event triggers a transition in the FSM. A test case is a sequence of user events and corresponds to a path in the FSM. As is the case for conventional software, FSMs for GUIs also have scaling problems, this is due to the large number of possible states and user events in modern GUIs. The new feedback-based technique has been used in a fully automatic end-to-end process for a specific type of GUI testing. The technique uses feedback from the execution of a seed test suite, which is generated automatically using an existing structural event-interaction graph model of the GUI. During its execution, the run-time effect of each GUI event on all other events pinpoints event-semantic interaction (ESI) relationships, which are used to automatically generate new test cases[12].

2.5 AUTOMATICALLY GENERATING TEST CASES FROM SYSTEM REQUIREMENTS MODELS

The researched work is to generate test cases from system requirement models. Software related accidents occurs when requirements are miscommunicated to the developers or are not delivered to them at all. Test cases generated directly from system requirements can be used to detect such errors. Safeware has developed a technique for automatically generating test cases from SpecTRM-RL models. SpecTRM-RL is a requirements-specification language that is based on a formal state machine Model. A SpecTRM-RL model describes system inputs, outputs, state values, and internal modes. A state value represents information inferred by the system regarding the current operating environment. Internal modes represent different collections of behaviour. For test case generation automatically it identifies a input sequence that satisfying the basic condition. The algorithm for determining input sequences for satisfying conditions starts by making a list of all the conditions in the model. Each of these conditions is initially marked "unsatisfied", indicating no input sequence has been identified to satisfy the condition. As input sequences are found to satisfy the conditions, they are each marked satisfied. These conditions and any other condition that must be satisfied at System Start are automatically satisfied. Software has developed algorithms to automatically generate test cases directly from SpecTRM-RL requirements models[3].

3 TEST SUITE GENERATION

Test suite are produced at the pre-processing stage. In this pre-processing state. Input data set are given as any java class file and it parses the methods and variables. It checks for the exceptions and control flow in the class file. It checks the initialisation and setup function for the java unit test cases are implemented and verified. Finite state

automata is used to encode the program structure and observe the common behaviour. It is done for finding the equivalent object states. The whole file setup are organised into four stages for test suite generation. They are setup, connect, authenticate, quit. These stages cover test suite that are build successfully or with any constraints. These constraints normally occurs with missing of previously described stages. For example consider any protocol stages. The initial parameters are covered in the setup functions and their establishment part are prescribed in the authenticate and connect functions and stages completing these are passed without any pointer exceptions and interfaces implementations. Consider the Figure 1 for test suite generation checking stages and later for type state mining prescribed below .

4 TYPE STATE MINING

In the Typestate mining different sets of objects and transitions are labelled with names. Labeling of mined Typestate is done to check object behavior at the dynamic loading of java class file. Processing of these instructions are done by observing programming behavior with their states.

Structure of the program is classified based on their initialisation and execution part and it is given as input for Typestate mining.To label the mined state it need to initialized with start and end variable. It is accomplished with transitions state to display the connected state and non covered state.

Consider Figure 1.the SMTP Protocol class for Typestate mining. First protocol state is initialised and checked for Typestate as 0. It then check for connected state with the objects referenced to the initialized part. If any of the missing transitions or branch coverage is not analyzed it returns the state as 1. It indicates the termination of the program state and looks for the initial state.

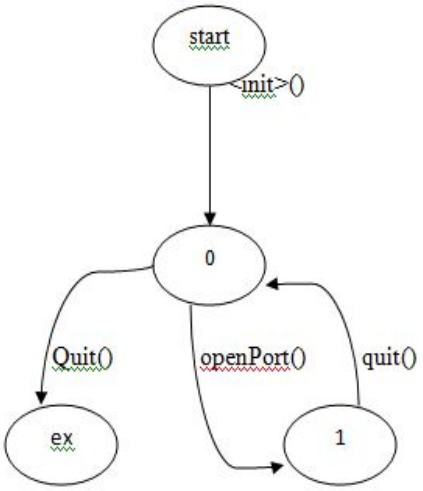

Figure 1.Type state for SMTP protocol

If any method in the Typestate causes an exception, it contains a transition from state to a special state ex labeled with method labelled name for branch coverage. If quit() is invoked from the initial state 0, this raises a NullPointerException. Transitions occurs when a method invoked on initialisation part changes its state. Object behaviour model are mined by the Type state miner from the execution of initially generated test suite. This stage express their connect functions that are covered in it. Initially it displays the INIT,QUIT,AUTHENTICATE states to 0. If the connect function not established it displays INIT state to 1 and others as 0. It shows the amount of transitions that covered in mining type state is larger than test suite. Consider the Table 1 and Table 2 for initial and mining type states transitions. The Table 1 describes about the initial test transitions from the input test class files. The occurrences of the class without method transitions are shown. In Table 2 mined type state with the mutants are described. It takes the transitions from all the prescribed states of class file with the references of states as prescribed earlier.

TABLE 1

Test suite generation statistics of SMTP protocol

Subject	State	Transitions	Exceptional conditions
SMTP PROTOCOL (mailing protocol)	131	61	11

TABLE 2

Test state mining statistics of SMTP protocol

Subject	Mutations	State	Transitions	Exp-condition
SMTP PROTOCOL	231	391	270	151

5 MUTATION OF TESTCASES

After generating the test suite and mining model for observed behaviour it is considered as initial model. Initial models then generates mutants for all methods that are executed in all states of initial model. Next initial model and all new model are combined and put into a common class under test. It allows to mine the model from each test and results in generating tests and for combining models in CUT.

The enriched Typestate checks the defects of wrong usage class. Typestate miner tracks the exception from the program. During the enriching of test cases it lacks behind the test cases generation for branch transitions.

Mutant generation starts by statically determining the set of methods that belong to the CUT or one of its super Types. For every such method m, TAUTOKO tries to generate mutations such that m is invoked in all states of the initial model. To invoke method m in states, TAUTOKO will either add an invocation of m, or suppress one or more existing method invocations. The choice of adding or deleting invocations depends on the number and Types of the parameters m expects. If m only requires a reference to the receiver object, TAUTOKO simply adds a new call to m right after a method call that caused a transition to s in the initial model

6 TEST CASES GENERATION

Test cases are generated by using finite state automata. For initial test suite test cases are generated by testing frame work. It uses branch coverage of the CUT as test objective. If the initial test suite is small, then more iteration takes to create an acceptable model for an initial test suite to the target class. From this initial test suite, it derive a Typestate automaton for generating and exploring the test cases.

To execute each method of the CUT in every abstract state, the set of methods are considered as inputs. First, a branch coverage test suite is generated to bootstrap the process, and an initial Typestate automaton is derived for this test suite. This automaton is traversed keeping track of the sequence of method calls that leads to the current state. For each state, it goes through the set of methods that have not been called in this state and generate a new test case that calls this method in the current state. Since it using a test generation tool for previously visited state than mutating existing tests, it applicable to any method, even it uses complex parameters. New test cases are executed, and a new model is learned from these executions and merged. This process is repeated until a fixed point is reached for finding missing transitions and over coming it.

7 EVALUATION

On comparing initial and enriched models in terms of quantitative and qualitative aspects, and their effectiveness in finding bugs when used as input to a static Typestate verifier for comparing the test case generation techniques. The metrics used are subject and enriching models. subjects describes the preset data set and their evaluation by checking with preset data from code snippets, open source tools,..Enriching models describes about enrich model for generation of test cases. Evaluation describes about the number of states and transitions. For generating and exploring type state automaton input parameters required to evaluate are class file, methods with basic type transitions. Table 3 shows the enriched models have more transitions with exceptional methods parameters.

This method initially investigates the visited test states and merge all the transitions. It invokes all the methods and prefix the generated type state automaton. It then recursively appends all the test class files. For enriching the models type state are investigated and performs the basic test suite generation of files with the suppress calls that are initially invoked from the class files. Figure 2 shows the enriched model of the SMTP protocol.

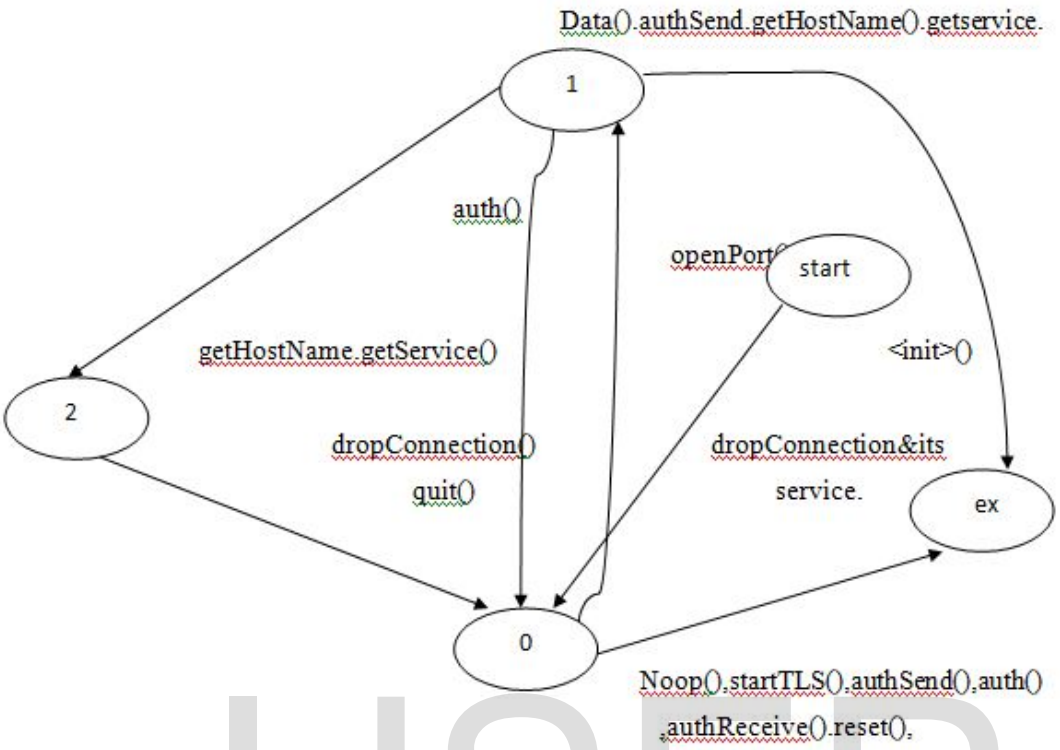

Fig 2 Enriched model of smtp protocol

TABLE 3

Enriched models with more number of transitions and exceptional not covered in the methods.

Statistics	State	Transitions	Exceptional conditions
Initial model of test suite generation	131	61	11
Enrich model	391	271	151
Exceptional Transitions of methods	1921	350	273

8 ENHANCEMENT

The specified technique is unable to handle methods with parameters that are never invoked by the program. Since it do not synthesize parameter values, TAUTOKO is unable to generate calls to these methods and unable to modify the test path. To overcome these limitations proposed new methods is to deal the first two limitations that are going to include a special test case which will deal the method calls via checking all the methods, through preprocessing. By adding a special case, it deals the method calls and method specific operations. It will check the methods and its variables, return variables, so if any of the

case above occur it will just call our new test case to deal the issue.

9 CONCLUSION

Dynamic specification mining depends on the observed executions, if enough test is not done it leads to incomplete state. To enrich the specification, TAUTOKO tool is used to generate test cases to cover all possible transitions between all observed states, and thus extracts additional states and transitions from their executions. This tool cannot invoke the methods during program call. To deal this limitation, behavioural analysis is done to generate
automatic test cases by integration of separate test cases for handling methods.

ACKNOWLEDGMENT

The authors would like to thank the Editor in chief,the Associate Editor and anonymous Referees for their comments.

REFERENCES

[1] Ashish Kumari, Noor Mohammad, Chetna," Specification Representation and Automatic Test Case Generation using System Model", IJSCE ISSN: 2231-2307, Volume-2,2012.

[2] Heinz Riener, Roderick Bloem, Gorschwin Fey," Test Case Generation fromMutants using Model Checking Techniques". The European Union (project DIAMOND, FP7-IST-4-248613),2009.

[3] Kenneth Kelley," Automated Test Case Generation from Correct and CompleteSystem Requirements Models", 978-1-4244-2622-5/09/$25.00-IEEEAC paper #1265, Version 3,2009.

[4] Pakinam N. Boghdady, Nagwa L. Badr, Mohamed Hashem and Mohamed F.Tolba(2011)," Test Case Generation and Test Data Extraction Techniques", International Journal of Electrical & Computer Sciences IJECS-IJENS Vol: 11 No: 03,2011.

[5] Pedro Flores and Yoonsik Cheon," PWiseGen: Generating Test Cases for Pairwise Testing Using Genetic Algorithms, IEEE International Conference on Computer Science and Automation Engineering CSAE,2011.

[6] Peter Frohlich and Johannes Link ," Automated Test Case Generation from Dynamic Models", Springer-Verlag Berlin Heidelber- Elisa Bertino (Ed.): ECOOP , LNCS 1850, pp.472–491,2000.

[7] Mark Gabel and Zhendong Su, "Symbolic Mining of Temporal Specifications",international conference of software engineering(ICSE)",2008.

[8] Sharon Shoham, Eran Yahav, Stephen J. Fink, and Marco Pistoia,"StaticSpecification Mining Using Automata-Based Abstractions", IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 34, NO. 5,2008.

[9] Stephan Weibleder and Bernd-Holger Schlingloff," Quality of Automatically Generated Test Cases based on OCL Expressions", international conference of software testing(ICST),2008.

[10] Valentin Dallmeier · Nikolai Knopp · Christoph Mallon · Sebastian Hack · Andreas Zeller(2010)," Generating Test Cases for Specification Mining", ACM 978-1-60558-823-0/10/07,2010.

[11] T. K. Wijayasiriwardhan P. G. Wijayarathna, and D. D. Karunarathna," AnAutomated Tool to Generate Test Cases for Performing Basis Path Testing", The International Conference on Advances in ICT for Emerging Regions - ICTer2011 : 095-101,2011.

[12] Xun Yuan, Member, IEEE, and Atif M Memon," Generating Event Sequence-Based Test Cases Using GUI Run-Time State Feedback", IEEE TRANSACTIONS ON SOFTWARE ENGINEERING,2010.

[13] Zhu Bin, Wang Anbao," Functional and User Interface Model for Generating Test Cases", IEEE/ACIS international conference,2012.

[14] V.Dallmeir, C.Lindig, A.Wasylkowski," Mining object behaviour with ADABU,"Proc.ICSE Workshop Dynamic Analysis,May 2006.

[15] http://www.st.cs.uni-saarland.de

[16] David Lo, Shahar Maoz "Specification mining of symbolic based scenario model", research collection school of information system notes.

Mahendran.N is doing ME(software engineering) in SNS college of technology, Coimbatore affiliated to Anna University Chennai, he also pursued B.tech(information technology) in sona college of technology, salem . His research area are software testing, cloud computing network security.

R. Kamalraj received his B.E. degree in Computer Science &Engineering from Bharathiyar University, Coimbatore, Tamil Nadu, INDIA in 2002, the M.E . degree in Computer Science & Engineering from Anna University, Chennai, Tamil Nadu, INDIA in 2009, and pursuing Ph.D. degree in Software Testing and Quality Management at Anna niversity of Technology, Coimbatore. He has published 7 papers in international journals and 1 paper in National Journal. He is having 9 years of teaching experience in 4 different engineering colleges. At present he is working as an Assistant Professor in the Department of Computer Science and Engineering at SNS College of Technology, Coimbatore. His research interests include Software Testing, Software Quality Management and Data Mining.

Professor Dr.S.Karthik is presently Professor & Dean in the Department of Computer Science & Engineering, SNS College of Technology, affiliated to Anna University- Coimbatore, Tamilnadu, India. He received the M.E degree from the Anna University Chennai and Ph.D degree from Ann University of Technology, Coimbatore. His research interests include network security, web services and wireless systems. In particular, he is currently working in a research group developing new Internet security architectures and active defense systems against DDoS attacks. Dr.S.Karthik published more than 35 papers in

refereed international journals and 25 papers in conferences and has been involved many international conferences as Technical Chair and tutorial presenter. He is an active member of IEEE, ISTE, IAENG, IACSIT and Indian Computer Society.