A Survey of UML-Based automatic Test Cases Generation for Software Testing

A.V.K.Shanthi, D.Parthiban
Department of Electronic Sciences
Sathyabama University,
Chennai, India,
avks_15@yahoo.com, parthi.best@hotmail.com

Dr.G.MohanKumar
Park College of Engineering,
Coimbatore, India,
gmohankumar68@yahoo.com.

Abstract—Software testing is relegated to one phase of the software development life cycle. Testing of software is a time-consuming activity which requires a great deal of planning and resources manually. To reduce time-consumption and to increase reliability researchers have tried to automate it. One of the important criteria of testing is test case generation. Test cases describe tests that need to be run on the program to verify that the program runs as expected. Automatic test case generation using model-based testing through use of a UML model of the system is being studied in this paper. This paper presents A survey of UML-based automatic test case generation for software testing that are found in the current literature. Problems in usage of certain techniques are identified. Areas that needed future research are presented.

Keywords— software testing, test cases, automated, UML-based testing, Activity diagram, Sequence diagram, Class diagram, Object diagram, Stat-Chart diagram.

I. INTRODUCTION

A very fundamental problem with software testing is that testing under all combinations of inputs and preconditions (initial state) is not feasible, even with a simple product this means that the number of defects in a software product can be very large and defects that occur infrequently are difficult to find in testing. More significantly, non-functional dimensions of quality (how it is supposed to be versus what it is supposed to do) usability, scalability, performance, compatibility, reliability—can be highly subjective; something that constitutes sufficient value to one person may be intolerable to another. A test case is a set of conditions or variables and inputs that are developed for a particular goal or objective to be achieved on a certain application to judge its capabilities or features. It might take more than one test case to determine the true functionality of the application being tested. Every requirement or objective to be achieved needs at least one test case.

A test case in software engineering is a set of conditions or variables under which a tester will determine whether an application or software system is working correctly or not. In some settings, an oracle could be a requirement or use case, while in others it could be a heuristic. Test cases are often referred to as test scripts, particularly when written. Written test cases are usually collected into test suites. A primary purpose for testing is to detect software failures so that defects may be uncovered and corrected. This is a non-trivial pursuit. Testing cannot establish that a product functions properly under all conditions but can only establish that it does not function properly under specific conditions.

II. LITERATURE SURVEY

Several researches have been successfully proposed test case generation for various software under various circumstances, mainly scenario-based, model based, path-oriented, goal-oriented and genetic approaches. Scenario-based techniques test cases based on concurrent approach with concurrence coverage criteria. Model based technique identify respective test case for the software with respect to the UML diagrams such as activity, state-chart, object diagram etc. Path-oriented testing based on static as well as dynamic control flow of the software. Static path testing done by symbolic execution, dynamic path testing based on the run time test of executing program. Goal-oriented techniques identify test cases covering a selected goal such as a statement or branch, irrespective of the path taken.

Many researchers and doctorates have been working in field of software testing in path toward generation test cases. Among them survey results in most of them use modeling language to generate test case. Since Unified Modeling Language (UML) is a standardized general-purpose modeling language in the field of software engineering. UML diagrams represent two different views of a system model static and dynamic. Static (or structural) view: emphasizes the static structure of the system using objects, attributes, operations and relationships. The structural view includes class diagrams and composite structure diagrams. Dynamic (or behavioral) view: emphasizes the dynamic behavior of the system by showing collaborations among objects and changes to the internal states of objects. This view includes sequence diagrams, activity diagrams and state machine diagrams.

Even though variety of approaches have been proposed, with the advent of modeling tools like Rational Rose, for a decade there has been constant research on generating test cases based on specifications and design models. For easy understanding, we have classified test case generation approaches mainly into two categories – Specification based test case generation and Model based test case generation [1].
In this paper survey has been done on the work carried out in current and past few years back on generating automatic test cases using UML Models.

A. SCENARIO-BASED TEST CASE GENERATION

Scenario-based test cases generation is based on concurrent application availed in used case studies. Following author employed themselves for generating test case in this scenario.

Baikutha Narayan Biswal [2] had presented paper on “A Novel Approach for Scenario-Based Test Case Generation”. In [2] they have proposed scenario-based testing, test scenarios are used for generating test cases. UML activity diagrams describe the realization of the operation in design phase and also support description of parallel activities and synchronization aspects involved in different activities perfectly. Where this paper deals with test adequacy criteria. Scenario testing works best for complex transactions or events, for studying end-to-end delivery of the benefits of the program, for exploring how the program will work in the hands of an experienced user, and for developing more persuasive variations of bugs found using other approaches. This implementation is carried out help of JAVA Swing and Rational Rose 7.0. ATM Application is used for this case study. Methodology involved in analyzing sequence and class diagram, Test unit definition, Search of setting and interaction categories, test case Construction. Advantage of this methodology is approach is that it handles the complicity of nested fork-join pair which is more often overlooked by other approaches. It overcomes the limitations of nested fork-join and loops.

Concurrent Application testing described by Chang-ai Sun [3] also employs UML Activity diagram for generating the test case. A paper on A Transformation-based Approach to Generating Scenario-oriented Test Cases from UML Activity Diagrams for Concurrent Applications promotes transformation-based approach to generating scenario-oriented test cases for testing concurrent applications modeled by UML Activity Diagrams. Concurrent behavior is nondeterministic its testing is more difficult than the testing of common control flows or data flows. When activity diagrams are used to model concurrent applications under test, a challenge of generating test cases from activity diagrams relies in their non-structural properties. The approach first transforms a UML activity diagram specification into an intermediate representation via a set of transformation rules. From the intermediate representation author constructed a set of test scenarios with respect to the given concurrence coverage criteria. Finally, derive a set of test cases from the constructed test scenarios. The approach employs transformation to resolve the nonstructural problem with activity diagrams, and can generate test cases on demand to satisfy a given concurrence coverage criteria and hence the number of the resulting test cases is controllable [3]. Scenario-oriented test cases from UML activity diagram specifications, with an emphasis on testing concurrent applications modeled using UML activity diagrams.

UML Activity Diagram Based Testing of Java Concurrent Programs for Data Race and Inconsistency presented by Bin Lei, Linzhang Wang, Xuandong Li illustrate the Data race occurs when multiple threads simultaneously access shared data without appropriate synchronization, and at least one is write. System with a data race is nondeterministic and may generate different outputs even with the same input, according to different interleaving of data access. We present a model-based approach for detecting data races in concurrent Java programs. We extend UML Activity diagrams with data operation tags, to model program behavior. Program under test (PUT) is instrumented according to the model. It is then executed with random test cases generated based on path analysis of the model [4]. Involve path based random test generation, tagging data operation in activity diagrams. A complex system often contains a set of scenarios. A data race may only occur when several scenarios appeared together at runtime.

Test Cases Generation from UML Activity Diagrams Presented by Hyungchoul Kim also based on concurrency in Activity Diagram i.e., concurrent system in which multiple Objects interact with each other. Proposes a method to generate test cases from UML activity diagrams that minimizes the number of test cases generated while deriving all practically useful test cases. Our method first builds an I/O explicit Activity Diagram from an ordinary UML activity diagram and then transforms it to a directed graph, from which test cases for the initial activity diagram are derived [7]. Method that is based on an I/O explicit Activity Diagram (IOAD) model, which is an abstraction model obtained from the fully expanded activity diagram by exposing only external inputs and outputs. This method avoids the state explosion problem that can occur when trying to derive a set of test cases with thorough coverage for concurrent system. By reducing the number of test cases for concurrent system testing while keeping all practically useful test cases, we can save cost and time in software development without compromising quality of the developed system.

Test Case Design Using Conditioned Slicing of Activity Diagram by Mitrabinda Ray, Soubhagyavati Sankar Barpanda, Durga Prasad Mohapatra presents conditioned slicing as a general slicing framework for test case generation from activity diagrams. Method first builds a flow dependence graph from an ordinary UML activity diagram and then applies conditioned slicing on a predicate node of the graph, to generate test cases. It minimizes the number of test cases generated while deriving all practically useful test cases. The effectiveness of a test case is based on how well the test covers and exercises the modeled behaviors. Proposed method satisfies high path coverage criterion [8]. Test scenarios are test requirements or test sequences (like possible execution Sequences of use cases, message sequences, event sequences etc) that must be satisfied or covered during testing. The exact value of test data (or test case) for which these test scenarios are tested is not addressed in many related works. Generate conditioned slices and slicing criteria based on the Flow Dependence Graph to make it applicable to Activity Diagram.
Construction of Flow Dependence Graph is static in our approach. They are continuously partitioning the input domain and slicing the FDG till getting a simple path.

B. MODEL BASED TEST CASE GENERATION

Model based test case generation equally challenging and also many researches involve achieving optimal set of test case. Following Analysis are based work evolved under model based test case generation.

Automatic test case generation using unified modeling language (UML) state diagrams by P. Samuel, R. Mall, A.K. Bothra published on basis of model based test case generation automatically. The approach, the control and data flow logic available in the UML state diagram to generate test data are exploited. The state machine graph is traversed and the conditional predicates on every transition are selected. Then these conditional predicates are transformed and function minimization technique is applied to generate test cases. The present test data generation scheme is fully automatic and the generated test cases satisfy transition path coverage criteria. The generated test cases can be used to test class as well as cluster-level state-dependent behaviors [5]. Test data generated fusing this approach is verified based on the path coverage. The step involved are, the first step is to select a predicate. In this, select a predicate on a transition from a UML state machine diagram. The next step is to transform the selected predicate to a predicate function. In the third step, generate test data corresponding to the transformed predicate function.

approach can handle change events, time events and transitions with guards, and achieves transition path coverage reduced the number of test cases that achieve transition path coverage by testing the borders determined by simple predicates.

Test Case Generation by means of UML Sequence Diagrams and Labeled Transition Systems by Emanuela G. Cartaxo, introduced UML Sequence diagram in field of test case generation. A feature is an increment of functionality, usually with a coherent purpose that is added on top of a basic system. Feature are usually developed and tested separately from the basic system as independent modules. The procedure is based on model-based testing techniques with test cases generated from UML sequence diagrams translated into Labeled Transition Systems (LTSs) [6]. The idea is to reuse sequence diagrams that are constructed by development teams to specify use cases with basic and alternative scenarios. Labeled Transitions Systems (LTSs) are used as an internal model to precisely represent the functional feature behavior. Proposed systematic procedure is appropriate to the reality of most mobile phone companies, since the test cases are derived from UML sequence diagrams and, in general, these diagrams are generated early in the development process. By generating test cases still in the development cycle, it is possible to decrease costs keeping the quality.

Model-based software development bases on setting up models of the system to be constructed. This approach has proved to be useful, because it allows developers to first elaborate the most important properties of the software before proceeding with the implementation. In Test Case Generation from UML State Machines presented by Dirk Seifert elaborates test cases include not only stimuli to trigger the system under test, they also include possible correct observations to automatically evaluate the test case execution. In contrast to classical Harel Statecharts, state machines behave asynchronously, which makes automatic test case generation a challenge. The TEAGER Tool Suite implements the automatic generation, execution and evaluation of test cases and proves the applicability of test approach [9]. With our approach UML state machines can be used in the quality assurance to serve as a specification for the desired reactive behavior of the system. It is possible to select relevant and interesting inputs for a test case and to calculate the possible correct observations for given inputs. They allow to automatically evaluating test executions which is in general a difficult and time consuming task. Applied approximation makes the generation process practical, whereat it is possible to control this process depending on the time and computation power to invest.

Automated-Generating Test Case Using UML Statecharts Diagrams by supaporn kansomkeat and wanchai rivepiboon experimented on the automatic testing technique to solve partially the testing process. This technique can automatically generate and select test cases from UML state chart diagrams. Firstly, transform this diagram into intermediate diagram, called Testing Flow Graph (TFG), explicitly identify flows of UML state chart diagrams and enhance for testing. Secondly, from TFG generate test case using the testing criteria that is the coverage of the state and transition of diagrams. Finally, the evaluation is performed using mutation analysis to assess the fault revealing power of our test cases [10]. Specification-based testing uses information derived from a specification to assist testing as well as to develop program. Testing activities consist of designing test cases that are a sequence of inputs, executing the program with test cases, and examining the results produced by this execution. Automatically generate test cases from UML specification with the aid of the Rational Software Corporation’s Rational Rose tool. Test cases are measured the effectiveness on the basis of their fault detection abilities. Results of simple test experiments are high effectiveness of the generated test cases. However, extensive experiments are needed to have more confidence of the testing technique and to compare it with other techniques in term of cost and effectiveness.

Test Case Generation Based on Use case and Sequence Diagram by Santosh Kumar Swain, Durga Prasad Mohapatra, and Rajib Mall illustrate Test cases are derived from analysis artifacts such as use cases, their corresponding sequence diagrams and constraints specified across all these artifacts. Construct Use case Dependency Graph (UDG) from use case diagram and Concurrent Control Flow Graph (CCFG) from corresponding sequence diagrams for test sequence generation. Focus testing on sequences of messages among objects of use case scenarios. Our testing strategy derives test cases using full predicate coverage criteria. Automation of test case generation is an important issue. Manual test design is
time consuming and error-prone. It is necessary to develop automatic tests design techniques. Therefore, with continually increasing software sizes, the issue of automatic design of test cases is assuming crucial importance. There are essentially two main approaches to automatic design of test cases. Proposed test case generation technique can be used for integration and system testing accommodating the object message and condition information associated with the use case scenarios. The test cases thus generated are suitable for detecting synchronization and dependency of use cases and messages, object interaction and operational faults.

C. GENETIC BASED TEST CASE GENERATION

Automatic Test Case Generation for UML Object diagrams using Genetic Algorithm presented by M. Prasanna and K.R. Chandran used to generate optimal test cases which also can be consider as data mining approach. Case study on object diagram for ATM application is illustrate. Automated generation of test cases in object oriented systems has been presented. The test cases are derived by analyzing the dynamic behavior of the objects due to internal and external stimuli. The scope of the paper has been limited to the object diagrams taken from the Unified Modeling Language model of the system. Genetic Algorithm’s tree crossover has been proposed to bring out all possible test cases of a given object diagram [12]. Experimental results show that it has the capability to reveal 80% fault in the Unit level and 88% fault in the integration level. We have viewed testing an application as traversing a path through the DFS (Depth First Search) for a binary tree to generate appropriate and accurate test cases. The mutation testing conducted has yielded 80.3% effectiveness in the actual testing process carried out with the generated test cases. Methodology is useful to generate test cases after the completion of the design phase and errors could be detected at an early stage in the software development life cycle.

III. DISCUSSION

To sum up all, there are various techniques available for generating test cases to satisfy test coverage as well as path coverage criteria. Scenario-Based test case generation mainly focused on concurrent process in only activity diagram. Model-Based test case generation focused on various state charts, sequence, object, use case diagrams to generate test case but fails produce optimal one. Genetic based test case generation produce an optimal on but still fault test cases are available. In recent trend Model-Based test case attracts many reaches by using some data mining concept to produce an automated optimal test case. By which minimum human and cost effort are utilized.

REFERENCES


A.V.K. Shanthi  
Research Scholar  
Sathyabama University  
avks_15@yahoo.com

D.Parthiban  
PG Student  
Sathyabama University  
parthi.best@hotmail.com

Dr. G. Mohan Kumar  
Principal  
Park College of Engineering  
gmohankumar68@yahoo.com