IMPACT OF UML TECHNIQUES IN TEST CASE GENERATION

Saru Dhir
Asst. Professor, ASET, Amity University, U.P., India, sarudhir@gmail.com

Abstract

UML is a standard language that used in business modeling for specifying, visualizing and constructing the software artifacts. UML provides the capability to enhance (explore) the static structure and dynamic behavior of a software system. Different UML strategies and techniques are implemented during the whole software development life cycle. This paper explains the UML2.0 testing for test case generation. In this paper, the focuses will be on effective use of UML techniques and test-case generation in order to make suitable executions.

Index Terms: Software Testing, UML specifications, Behavioral Diagrams, Structure Diagrams.

1. INTRODUCTION

Software testing is a technique for software verification and validation by creating different test cases whose execution will detect possible errors. Specification of UML is growing day by day. UML has achieved a good deal of attention from the software design and development communities. To execute the real time applications new timing diagrams are added in UML diagrams.

UML is a graphical language which partitions into different logical sub packages: Structural Diagram and Behavioral Diagram. Structural packages provide the basic infrastructure for describing the static structure of the system. This infrastructure includes: Class Diagram, Implementation Diagram and Object Diagram.

Behavioral Package elements provide the linguistic elements for modeling the dynamic behavior of the system. This infrastructure include: Use Case Diagrams, Interaction Diagrams, Activity Diagrams, and State Diagrams.

The rest of the paper is organized as follows: Section 2 defines the concepts of UML testing. Section 3 defines the tools and techniques based on analysis through UML structure Diagram. Section 4 describes the tools and techniques for generating test cases through Behavioral Diagrams. The paper concludes with section 5.

2. UML TESTING CONCEPTS:

UML 2.0 testing provides concepts that target both the pragmatic and systematic development of concise test specification and test models [1]. Different concepts are used in UML 2.0: Test Architecture, Test Behavior and Testing Time.

Test Architecture specifies the structural aspects of a test system including test context, the System under Test (SUT), Test components and the scheduler. Test Behavior specifies the actions and evaluations that are necessary to describe the aim of a test. Behavior diagrams specify the behavior of the system. Some of the example of behavior diagrams is Activity diagrams, state machines and interaction diagrams that can be used to define test stimuli, test control, coordination’s and actions. Test Time concepts are specified to complete the concepts needed for test modeling. The Timing concepts included in UML 2.0 are Timers and Time zones, to manipulate and control the test behavior and to group components within a distributed system.

For Test implementation and execution environment to mapping are provided: JUNIT is an open source unit testing framework, is implemented in Java. TTCN-3 is widely accepted as a standard for test system development in the telecommunication and in data communication area [2].

In this approach focus is on techniques by analyzing the UML Structural and Behavioral Diagrams. I focus on the methods or techniques used to generate test cases from UML activity diagrams to minimize the number of test cases.
3. ANALYSIS THROUGH UML STRUCTURE DIAGRAMS

Structure model provide an explanation of the system, i.e. what should be there inside the system that is being modeled. Some of the basic examples of structure diagrams are: Class Diagrams, Object Diagrams and Deployment Diagrams as shown in Fig 1.

First the code instrumentation is executed under test. Instrumented code is tracked by number of times and provides the information used for reducing redundant test cases. In code, it will make sure that all the classes are being accessed during execution. After the coding part, test cases are generated for object diagrams and class diagrams. Features of all classes such as method names, their parameters and return types etc. are extracted and accessing the object diagrams. All these extracted features and different test cases based on the extracted information’s are generated. In the class diagrams, we can extract the relationship of the class with other classes. After running test cases generated, execution trace is measured with help of code. In other words, calculate the number of time a method is called and analyzing the control flow of the application. Different test cases are generated by object diagrams and class diagrams. Now comparing between both of these diagrams, we can observe which will be redundant test cases and how can we remove it.

4. ANALYSIS THROUGH BEHAVIORAL DIAGRAMS

UML has now become the de facto standard for object oriented modeling and design [3]. UML models are an important source of information for test case design, which if satisfactorily exploited, can go a long way in reducing testing cost and effort and at the same time improve software quality [4]. In UML, use cases can be represented by using behavioral diagrams such as interaction, activity and state machine diagrams.

During software development process test cases are generated by using UML diagrams. Through this functionality we identify the bugs and the problems earlier and it will save more time. Test cases are generated and reducing irrelevant test cases by following the paradigm of classes and objects. Different efficient approaches are applied on different applications such as Java application in which test are generated and implemented effectively.
events during interaction between objects. Twng D.Trong [7] proposed a systematic approach to testing design models described by UML class Diagrams, sequence diagrams and activity diagrams and also test adequacy criteria for those diagrams.

To generate test cases, a message sequence path criteria is used from sequence diagrams. For each sequence diagram, a test case $T$ is assigned such that when the software is executed using $T$, the software that implements the message sequence path of the sequence diagram must be executed [5].

**Activity diagram** focuses on the control flow among objects and it is the best way to show the conditional flows of execution. In sequence diagram, we can only specify the message paths. But if sequence and activity both are used, message as well as activity path can be covered which is called Message-Activity-Path. Message-Activity-Path is the super-set which ensures Message-Path is the subset. Message-Path coverage is a stronger testing technique compared to message coverage [5].

There are different techniques developed, that generate test cases from UML activity diagrams. Test cases were generated from UML activity diagrams, which modify Depth First Algorithm (DFS) for automated generation [6], where basic paths can be found by DFS algorithm.

Input-Output explicit Activity diagrams (IOAD) introduced, which is converted to a directed graph for extraction of test scenarios and test cases from IOAD Model. IOAD shows the external inputs to and external outputs. To minimize the number of test cases from an IOAD model, two principles are adopted: First one is “the principle of Black Box Testing” and the Second one is “the single stimulus principle”.

**State Chart Diagrams** specifies all possible states of objects in a class and it shows a state machine consisting of states, transitions, events and activities.

Test Cases are automatically generated with Rational Rose Tool. State Chart Diagrams created from this tool and transformed to Testing Flow Graph. Testing Flow Graph is simple structure diagram and it reduces the complexity of UML Statechart Diagrams.

In Statechart, states are represented in rounded rectangles and transitions are indicated by arrows.

As we discussed, Statechart Diagrams transform to a hierarchy structure testing Flow Graph before being able to generate test cases. Fig.5 shows the graph consisting of transitions and nodes. The state name in Statechart is replaced by s-node and each group of events represents transitions of this state.

In Fig. 4 the state A is s-node from the Statechart is State Name where g-node is source node of transitions. A1 and A2 are two g-nodes for s-transitions $e_1$ and $e_2$ respectively.

5. CONCLUSION AND FUTURE SCOPE

In this paper I have provided a brief explanation of the testing UML 2.0. The UML 2.0 provides the different techniques and tools for generating the test cases. In this paper I focused on the different techniques used for generating test cases that minimize the number of test cases in Behavioral and Structural diagrams. Behavioral elements from UML 2.0 can be used to specify the dynamic nature of test cases. These include interaction, state chart and activity diagrams. Structural
elements includes: class diagram, object diagram and implementation diagrams.

In the future, I plan to develop an automated tool for generating test cases. I also plan to implement the techniques for improving the performance of test case generation step and measuring the data coverage during the improvement of test data creation and utilization of data during execution of tests.

REFERENCES


