 IMPORTANCE OF UNIFIED MODELLING LANGUAGE FOR TEST CASE GENERATION IN SOFTWARE TESTING

Bipin Pandey  
Department of CSE & IT  
Vyas Institute of Engineering and Technology, Jodhpur  
Rajasthan, India  
bipinpandey86@gmail.com

Rituraj Jain  
Department of CSE & IT  
Vyas Institute of Engineering and Technology, Jodhpur  
Rajasthan, India  
jainrituraj@yahoo.com

Abstract
Manual testing usually required high number of resources, as well as it incurred high cost and time. 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 which describes tests that need to be run on the program to verify that the program runs as expected. In many software development organizations, the cost of testing can account for more than 40% of the total development cost for a software system. Researchers are exploring the possibilities of using UML Diagrams to finding good set of test cases. This paper aim to exploring few of the approached based on the UML Diagram to generate test cases for effective software testing.

Keywords: Activity Diagram, Class Diagram, Collaboration Diagram, Object Diagram, Sequence Diagram, Software Testing, State Chart Diagram, Unified Modeling Language, Use Case Diagram.

1. Introduction

Many software products fail in some or the other ways, just because of improper testing. Software testing is one of the most important phase of the software development cycle to enhance confidence in development team as well as customer about the developed software. As we know that errors commence at the early stages of the software development will add more numbers of errors in the later phases. If the testing will follow from the initial phases of the development process, number of errors can be minimized [1], [2]. In the recent era of software development process Software developers are using the Unified Modeling Language (UML) and other modeling tools for the design and implementation of their products. These modeling tools can be one of best source to design the test cases for software testing. Designing tests based on UML diagrams will give high quality results, a more robust system. UML diagrams can be used for automated testing. This would enable the developers and testers to test even at the early stages of software development. In this paper, we study the work done in the area of software testing based on the UML diagram [3] [4].

2. Unified Modeling Language

UML used during software development by specify, visualize, and document models of software systems, including their structure and design [5]. The UML is typically used with the support of a suitable CASE tool, to define the requirements, components and their interactions in the proposed software system [6].

UML is one of the best modeling tools used for defining the structure of the system as well as to manage large and complex system. UML offers a standard way to visualize a system's architectural blueprints, including elements such as: activities, actors, business processes, database schemas, (logical) components, programming language statements, and reusable software components. It can be used with all processes, throughout the software development life cycle, and across different implementation technologies. UML diagrams represent two different views of a system model which are Static (or structural) view and Dynamic (or behavioral) view. The structural view includes class diagrams and composite structure diagrams and the dynamic view includes sequence diagrams, activity diagrams and state machine diagrams [7].
3. Use Case Diagram Based Testing

Use Case Testing is a functional black box testing technique that helps testers to identify test scenarios that exercise the whole system on each transaction basis from start to finish. A use case may be defined as a sequence of actions a system executes to yield an observable result of value to its actors. In the Unified Modeling Language (UML), the use cases are graphically represented in a diagram, which also shows the relationships among them. These relationships determine the diagram structure [8].

Basic Characteristics of Use Case Testing are capturing the interactions between 'actors' and the 'system', where Actors' represents user and their interactions that each user takes part into. Test cases based on use cases and are referred as scenarios with capability to identify gaps in the system which would not be found by testing individual components in isolation [9].

Adriana Carniello et al. [8] introduced a novel set of testing criteria based on the structure of use case diagrams. The structure of the use case diagram is defined by the relationships it contains, namely, association, include and extend relationships. They designed Use Case Tester (UCT) tool determines the new criteria test requirements, emulates the use cases behavior and analyzes the test coverage with respect to the criteria for a given use case diagram. As UCT emulates the behavior of use cases it can be utilized to simulate tests cases execution as well to validate requirements with the final user. UCT’s novelty resides in this aspect, particularly useful in requirements engineering. They defined testing criteria based on the relationships which are All-Associations-Inclusions-Extensions Criterion (c1) and All-extended-combinations Criterion (c2). Criteria c1 suggest that given a test set T and a use case diagram D, for each use case in D extended by at least two other use cases, T must cause all the combinations of exercising and non-exercising the extend relationships to be exercised at least once. Criteria c1 suggest that given a test set T and a use case diagram D, T must cause each association, include and extend relationship in D to be exercised at least once.

M. Katara et al. [10] defend a method in which use cases are converted into sequences of so called action words, which are simple translation from the events listed in the use cases. The action word sequences are then processed by test coverage language, which will be used in a simple algorithm for test generation. The selection of a particular test coverage language requires the fulfillment of criteria, as outlined in [11].

C. Nebut et al. [12] proposed an approach to generate tests cases from a use case in the context of object-oriented embedded software. They defined a simulation model of the use cases which is used to explicitly build a model of all the valid paths through use cases, and to extract relevant paths from it using coverage criteria. These relevant paths are ‘test objectives’. Here the test cases are generated in two steps: The derivation of test objectives from the use cases constitutes the first step. Their Use Case Transition Systems (UCTS) describes following criteria for generation of test objectives:

- All Edges criterion (AE)
- All Vertices criterion (AV)
- All Instantiated Use Cases criterion (AIUC)
- All Vertices and All Instantiated Use Cases criterion (AV-AIUC)
- All Precondition Terms criterion (APT)
- Robustness criterion

The second step involves translation of test objectives into test scenarios. Test scenarios may differ from the test cases in the fact that the test cases can be applied directly with a test driver, whereas the test scenarios may still be incomplete. The test scenarios contain the main messages exchanged between the tester and the system under test. Each use case is documented by its contracts and scenarios illustrating how the system has to be stimulated by the actors in order to perform the use case and how the system should react to the stimulation. We assume that these scenarios are expressed with UML sequence diagrams.

4. Class Diagram Based Testing

The class diagram is a static diagram. The purpose of a class diagram is to depict the classes within a model. Class diagram is not only used for visualizing, describing and documenting different aspects of a system but also for constructing executable code of the software application. The class diagram shows a collection of classes, interfaces, associations, collaborations and constraints. It is also known as a structural diagram. The basic unit of testing an object-oriented application is a class, and class testing work has mostly centered on functional testing [3].

M. Prasanna et al. [3] proposed an approach in which they are extracting the all data and member functions from the designed class diagram for real system accepted by used. By identifying and executing the first method to be executed they
applied Use-pair method for data variables until Use-pair is not getting null. Finally test sequences are generated. They conclude that this approach will help to commence the testing process early in the software development cycle. Requirements traceability can be done by this approach as the models form the basic building blocks of system design. The generated test case from data flow approach can further considered for the validation.

5. Object Diagram Based Testing

Object diagrams describe the static structure of a system at a particular time and they are used to test the accuracy of class diagrams. Object diagrams use a subset of the elements of a class diagram in order to emphasize the relationship between instances of classes at some point in time [13].

M. Prasanna et al. [14] prepare an approach in which an object diagram and mapped it to a tree with root node and child nodes. Then they apply Genetic Algorithm’s cross over operator which yields in new generation of trees and then convert them to into binary trees. Depth First Search (DFS) technique is applied on the binary trees which results in test case set. All the valid, invalid and termination sequences of the application can be obtained using traversing using DFS. They conclude that our 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.

6. Sequence Diagram Based Testing

A sequence diagram shows an interaction, which represents the sequence of messages between instances of classes, components, subsystems, or actors i.e. it is a graphical view of the exchange of messages between objects and actors for a use case, the execution of an operation, or an interaction between classes. It conveys the same kind of information as a communication diagram, except that it concentrates on the chronology of messages passing between the objects in place of their structure. Sequence diagram emphasizes on time sequence of messages. Sequence diagrams typically are associated with use case realizations in the Logical View of the system under development. A sequence diagram shows, as parallel vertical lines (lifelines), different processes or objects that live simultaneously, and, as horizontal arrows, the messages exchanged between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner.

A.V.K. Shanthi et al. [2] first extract the necessary information from the diagram for generating the test case from the sequence diagram. Based on the extracted information, a Sequence Dependency Table (SDT) is generated. With the help of SDT test path are generated, by applying the GA most prioritized test case are generated.

Ashalatha Nayak et al. [15] propose an approach of synthesizing test data by using the attributes and constraint information associated with the model elements such as class diagrams, sequence diagrams and Object Constraint Language (OCL) constraints. By using this derived information they augment the sequence diagram and then map it onto a structured composite graph (SCG) from which the test specifications are generated. For each test specification, they follow a constraint solving system to generate test data. The test data generated using three steps: deriving constraints for the specified scenario, solving the constraints along the scenario and then generating test data for finding test input to the variables involved in the scenario.

7. Collaboration Diagram Based Testing

A collaboration diagram describes interactions among objects in terms of sequenced messages. Collaboration diagram emphasizes on the structural organization of the objects that send and receive messages. Collaboration diagrams represent a combination of information taken from class, sequence, and use case diagrams describing the static structure and dynamic behavior of a system.

Aynur Abdurazik et al. [16] presented an approach to generating tests data to check aspects of the software that are represented by the collaboration diagrams. Collaboration diagrams provide the following six pieces of information viz are: objects, sequences of operation, the semantics of an operation, imported operation from other classes, the communication pattern of objects and the execution characteristics of objects. This information must be preserved during the transformation of a specification into an implementation. It is possible and necessary to formulate test requirements from the above points, and generate tests from the collaboration diagrams. Instrumentation will ensure that tests satisfy the formal testing criteria developed in this research, and also help ensure traceability from the design artifacts to the code.
8. State Chart Based Testing

A state chart diagram is a view of a state machine that models the changing behavior of a state. This diagram models the dynamic flow of control from state to state within a system. Statechart diagrams show the various states that an object goes through, as well as the events that cause a transition from one state to another.

Jasmine Minj et al. [17] propose a technique to generate test cases from UML State diagram that is based on path oriented approach. Genetic algorithm is used with stack based approach to get the optimized feasible test cases. Effectiveness of test cases generated from UML Statechart is measured by state coverage, transition coverage and transition pair coverage criteria. They present path-oriented test data generation which aims to generate feasible test cases that covers every possible path in the program. Intermediate graphs are created using the Statechart from which predicates will be extracted and represented in the form of binary bits which is taken as chromosome. Based on predicates, traverse the graph using DFS for test sequence generation. Cost of each path is calculated using McCabe’s formula of cyclomatic complexity formula. Fitness function is calculated by the cost of path and stack weight for each path. Selection is done using roulette wheel method and the individual probability is calculated based on the fitness of the individual. Then crossover operation will be applied and recombines the selected pairs of individuals. Bits are mutated which helps in introducing diversity into the genetic pool. It adds new individuals randomly to the population and thereby avoids solution being stuck in the local optima. Their results show that generated test cases using these methods are effective, efficient and optimized.

M. Riebisch et al. [18] proposed a method for generating system level test cases based on the used case models which are refined by state diagram. Their methods comprises of five basic phase which are refinement of use cases, transformation of use case model to state diagram, generation of usage graph from state diagram, transformation of usage graph to usage model and finally generation of test cases from usage models. The whole process will be proposed like this: They first created use case templates which are then transformed into state diagrams. The introduced use case template has been enhanced to support the systematic derivation of a state diagram usage specification and in this way, the independent usage profiles can be refined independently, retaining and building upon the common specification derived during the previous steps. Then top-level state diagrams are transformed into usage graphs. Transitions between states are labeled with the user action causing the transition to be taken. To obtain usage models the probability distribution of the expected use of the software with respect to the usage profile represented by the usage graph must be determined. Based on a usage model, it is possible to produce data that can be used throughout the project’s life cycle for test planning and to derive statistically valid samples of test cases.

9. Activity Diagram Based Testing

The Activity Diagram can help to describe the flow of control of the target system, such as the exploring complex business rules and operations, describing the use case also the business process. Activity diagrams show the workflow from a start point to the finish point detailing the many decision paths that exist in the progression of events contained in the activity [19]. The activity can be described as an operation of the system. So the control flow is drawn from one operation to another. This flow can be sequential, branched or concurrent. Activity diagrams deals with all type of flow control by using different elements like fork, join etc.

Debasish Kundu [20] proposed approach to generate test cases from an activity diagram by using following three steps: augmenting the activity diagram with necessary test information, converting the activity diagram into an activity graph, and generating test cases from the activity graph. Their approach is significant due to following reasons: it is capable to detect more faults like faults in loop, synchronization faults than the existing approaches; test case generated may help to identify location of a fault in the implementation, thus reducing testing effort; it inspires developer to improve design quality, find faults in the implementation early, and reduce software development time; and finally it is possible to build an automatic tool which will reduce cost of software development and improve quality of the software.

Dong Xu et al. [21] propose an automated approach using adaptive agent’s exploration to directly generate test scenarios from the activity diagrams. When using the proposed agents to search an activity diagram, special attentions need to be paid to two types of nodes, final nodes and branch nodes, in an activity diagram. They define a scenario tree as an interim storing structure. The initial node of an activity diagram is the root of the tree while each tree node denotes the elements of the activity diagram.
such as the action states, the forks, the joins, the branches, the merges, the guard conditions, and the final states. Test scenarios can then be derived from the scenario tree by traversing the tree.

SG Shukla et al. [22] proposed an approach for generating the test cases from activity graphs. Activity graph is a directed graph and transformed form of activity diagram where each node represents a construct and each edge represents the flow. Activity diagram are augmented with the necessary test information. Test cases generated from the activity graph by following the proposed test coverage criterion. They proposed a test coverage criterion, called activity path coverage criterion. They use this coverage criterion for both loop testing and concurrency among activities of activity diagrams. Their work considers concurrent activity path for an activity diagram that contains concurrent activities. For effective testing with limited resource and time, they test only relative sequence of the concurrent and non concurrent activities that is, set of precedence relations exist among these activities. The generation of entire set of concurrent activity paths by finding representative concurrent activity path from activity graph. This will make the task of test case generation process easier and hence, reduce testing effort.

Chen Mingsong et al. [23] proposed an automatic test case generation approach where they used activity diagrams as design specifications. Instead of deriving test cases from the activity diagram directly, an indirect approach is suggested in their approach which selects the test cases from the set of the randomly generated test case according to a given activity diagram. Then, by running the program with the generated test cases, they get the corresponding program execution traces. Last, by comparing these traces with the given activity diagram according to the specific coverage criteria, they get a reduced test case set which meets the test adequacy criteria. The approach can also be used to check the consistency between the program execution traces and the behavior of UML activity diagrams. The approach can also be used to check the consistency between the program execution traces and the behavior of UML activity diagrams.

P. N. Boghdady et al. [24] proposes an automated approach for generating test cases from the activity diagram. The proposed model automatically creates a table called Activity Dependency Table (ADT) and then uses it to create a directed graph called Activity Dependency Graph (ADG). The ADT is constructed in a detailed form that makes the generated ADG covers all the functionalities in the activity diagram. Finally the ADG with the ADT are used to generate the final test cases. Their proposed model includes validation of the generated test cases during the generation process to ensure their coverage and efficiency. The generated test cases meet a hybrid coverage criterion in addition to their form which enables using them in system, regression as well as integration testing. Their model saves time and effort besides, increases the quality of generated test cases.

10. Conclusion

In this paper, different models and approaches design to generate test cases or testing approached based on the Unified Modeling Languages has been described. Each approach has advantages as well as disadvantages.

11. References