International Journal of Engineering Research & Management Technology

May- 2015 Volume 2, Issue-3

Email: editor@ijermt.org www.ijermt.org

An Advance Technique of Test Case Generation Based on UML Diagrams

Mohd. Muneer Siddiqui
M.Tech Scholar
Computer Science and Engineering
Al-Falah School of Engineering and Technology
Dhouj,Faridabad,Haryana

Sher Jung Khan
Assistant Professor
Computer Science and Engineering
Al-Falah School of Engineering and Technology
Dhouj,Faridabad,Haryana

ABSTRACT:
Software testing strategy is one of the much more required part of software development life cycle (SDLC) for successful run of software / computer applications. The collapse of software without accurate testing is very large. Hence software testing is required more attention to keep away from the arbitrary failure or crashes of software product in SDLC. UML Diagrams are the basic models used to develop test paths from intermediary graphs generated automatically using graph coverage techniques. There are many different kinds of coverage criteria. For example - control flow, focused on data flow, boundary values, or transition sequences. In this paper, we will present new approaches, e.g. to combine coverage criteria and generation of test paths manually as well as automatically using tools based on Chinese postman and prefix based algorithms. Testing can be divided into two types such as White Box testing and Black Box testing. White box testing is completed through detail analysis of program structure where as black box testing deals with specification and design document i.e. without any details. Special attractions are needed to look into these qualities while testing is found out.UML supports object-oriented technology, which is widely used to describe the analysis and design specifications of software progress. UML models are an important source of information for test case design.

INDEX TERMS: Test data, UML diagrams, Model based testing, Object oriented technology

INTRODUCTION:

Software testing technique performs an important role during the design, development and release of software product. We have proposed an idea to test software at early stage with the help of Unified Modeling Language (UML). The paper summarized in five sections. Section-2 explained the Software Testing. Section-3 Described the Unified Modeling Language, in section-4 described a Object Oriented Technology and Software Testing, section-5 we have mentioned Test Path Generation Algorithm, and the section-6 we have written Conclusion and Future Work. Testing is the most important branch of the software development process in which we want to verify if a product satisfies the given requirements. It is a important process with many main parts. As products become progressively more complex, the process has become very broad and time consuming. The subject of test case generation is becoming more and more popular and because test case design and execution are time and resource consuming, it is understandable that automatic test case generation constitutes an important topic. Test cases can be generated from code, graphs, formal specifications and different models. Testing from models are also known as model based testing (MBT). MBT is a testing method that usually facilitates the automation of a test case creation using either models or properties. The work on this paper is focused on functional model-based testing. Functional testing is related with verifying of the system under test (SUT) with a Software Requirement Specification (SRS). A functional test detects a failure if the practical and the particular behavior of the SUT do not match. Model-based testing is about using

models as specifications. Test cases are frequently generated based on program source code. Another approach is to generate test cases from condition developed using formalisms such as UML models. In this approach, test cases are developed during study or design stage itself, preferably through the short level design stage. Each test path is then changed into a test. If we can generate fewer and shorter test paths, the charge of testing can be reduced.

SOFTWARE TESTING:

Testing of software product is the process of exercising a program with fine designed input data with the intent of observing failures. In other words, "Testing is the process of executing program with the purpose of finding errors". Testing identifies faults, whose exclusion increases the software quality by growing the software's potential reliability. Testing also measures the software quality in terms of its capability for achieving accuracy, maintainability, reusability, consistency, usability and testability.
Different testing techniques reveal different quality aspects of a software system, and there are two major categories of testing techniques such as functional testing and structural testing.

A. FUNCTIONAL TESTING:

The software program or system under test (SUT) is considered as a "black box testing". The selection of test cases for functional testing is based on the requirements or design conditions of the software entity under test. Expected results sometimes are called test oracles, which contain requirement/design specifications, hand calculated values, and virtual results. An exterior activity of the software entity is the main attraction of functional testing.

B. STRUCTURAL TESTING:

The software entity is measured as a "white box testing". The choice of test cases is based on the performance of the software entity. The main focal point of such test cases is to cause the execution of specific spots in the software entity, such as specific statements, program branches or paths. The estimated results are evaluated on a set of coverage criteria like branch coverage, path coverage, and data-flow coverage. Interior structure of the software entity is the core focus of structural testing.

UNIFIED MODELING LANGUAGE:

The UML is a diagram modeling language and used for visualize, construct, specify and document the artifacts of a software system. The major area of UML views: Dynamic, Structural and Model Management. Dynamic view considered activity diagram, state chart diagram, sequence diagram and collaboration diagram. Structural view includes class diagram, use case diagram, deployment diagrams and component diagrams. Model management view includes class diagram. Class diagrams - class structure and associations, component diagrams- map classes to software unit, deployment diagrams - physical system structure. The behavioral elements includes: Use Case Diagrams - functionality as seen by actors, Interaction Diagrams - object message sequences, Activity Diagrams - business / software processes, State Diagrams - object state transition behavior

A. Software Testing Based on UML:

StefaniaGnesi, propose a proper conformance testing relation for input-enabled transition systems with transitions labeled by input/output-pairs (IOLTSs). IOLTSs provide a suitable semantic model for a behavioral subset of UMLSCs. They had provided an algorithm which, for a UMLSC specification and the alphabet of implementations, generates a test suite. The algorithm is proven exhaustive and sound w.r.t. the conformance relation. Ad-Hoc /Random Techniques: These techniques construct regression test suite by selecting randomly test cases from unique test suite. Randomly rerunning test cases do not deal with the coverage of affected portions and may not find the most severe faults. Retest-All Techniques: These techniques rerun the entire

original test suites to ensure that modifications have not regress the software functionality, but this requires enough time or resources to rerun the entire test suites.

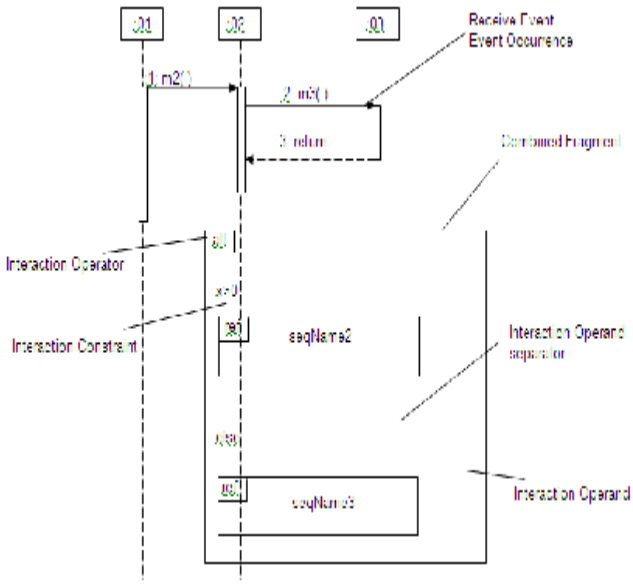
WaldemarPires had proposed a fully computerized technique to perform conformance checking of Java implementations beside UML class diagrams. In their approach, reused the Design Wizard Java API that allows us to write design rules as J Unit tests, i.e., to write them as code directly in the programming language. Several experiments on simple scenarios, Simple designs involving classes, associations, inheritance have been checked. Advantage of approach lies in the fact that we automatically generate design tests from UML class diagrams to Java code that play the dual role of design test and implementation language. UML diagrams, specifically class diagrams, are widely used in various software development processes; therefore, approach may be used in these processes to check the conformance between design and implementation. Kim proposes a method to produce test cases from UML activity diagrams that minimizes the number of test cases generated while deriving all practically useful test cases. This 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.

B. RELATED UML DIAGRAMS:

UML is a language for visualizing, specifying, constructing and documenting the artifacts of software systems. UML provides a selection of diagrams that can be used to current dissimilar views of an object-oriented system at different stages of the SDLC. Software testing techniques are based on the UML occupy the source of test requirements and coverage criteria from these UML diagrams. These diagrams are Use case, activity, sequence, collaboration (also called communication)and interaction diagrams. Sequence and communication diagrams give message level facts of a system, which are wanted for Control Flow Analysis (CFA). UML diagrams can be divided into three broad categories: structural, behavioral and interaction diagrams. The UML structural diagrams are used to model the static aspects of the dissimilar elements in the system, whereas behavioral diagrams focus on the dynamic aspects of the system. Our test generation method uses information present in two diagrams, which are use case and sequence diagrams. A use case comprises special possible sequences of interactions between the computer and the user. Each specific sequence of interactions in au se case is called a scenario. A use case is an abstraction of a system response to external inputs. It accomplishes job that is very important from user's point of view. Use case diagrams do not present architectural models, user-interface models or workflow models. A sequence diagram shows a set of objects and the sequence of messages exchanged between them. In a sequence diagram the importance is on the time ordering of the messages. The basic testing practice applied to sequence diagrams; all end-to-end paths should be known and exercised. This is equivalent to the requirement to classify and exercise all transitive relations formed by the variation of client-sends-message to server.

C. UML 2.0 SEQUENCE DIAGRAMS:

Sequence diagrams are important UML artifacts for modeling the behavioral aspects of a system. The diagrams are mainly well-suited for object-oriented software, where they represent the flow of control through object interactions. A sequence diagram defines a set of interacting objects and these quence of messages exchanged between them. The diagram may also include other information about the flow of control through the interaction, such as conditions and iteration or state-dependent behavior. Some of the new features are illustrated with an example given in Figure.



An interaction is a sequence of messages passed between objects to accomplish a particular task. An Interaction Occurrence is a symbol that refers to an interaction that is used within another interaction or con text. Seqd2 and seqd3 are two interaction occurrences of Seqd1 which refer to sequence diagrams seqd2 andseqd3. Event Occurrences represent moments in time to which actions are associated. It is the basic semantic unit of interactions. Event occurrences are ordered along a lifeline. A message has two types of Event Occurrences: Send Event and Receive Event. The Send Event is at the base (source) of the message arrow, while Receive Event is at the arrow head of the message arrow.

OBJECT-ORIENTED TECHNOLOGY AND SOFTWARE TESTING:

It is broadly received that the object-oriented paradigm will very much increase the software reusability, reliability, extendibility and inter-operability. Object-oriented software testing (OOST) is insignificant software quality assurance activity to ensure that the benefits of object-oriented (O-O) programming will be realized. Object-oriented software testing has to deal with new problems introduced by the object-oriented features such as encapsulation, inheritance, polymorphism, and dynamic binding. Below, we talk about different type levels of testing associated with object-oriented programs.

A. Intra-method testing:

Tests designed for individual methods. This is the same to unit testing of straight programs.

B. Inter-method testing:

Tests are constructed for pairs of method within the same class. In other words, tests are considered to test interactions of the methods.

C. Intra-class testing:

Tests are constructed for a single entire class, usually as sequences of calls to methods inside the class.

D. Inter-class testing:

It is use to test more than one class at the same time. It is just similar to integration testing.

The first three testing are of unit testing type, whereas inter-class testing is a type of integration testing. The overall approach for object-oriented software testing is equal to the one applied for conventional software testing but differs in the approach it uses.

As classes are integrated into an object-oriented architecture, the system as a whole is tested to ensure that errors in requirements are uncovered.

TEST PATH GENERATION ALGORITHM;

The explanation of the conceptual test case generation algorithm, whose purpose is the creation of a conceptual test case with conceptual information about inputs, is presented here. The algorithm starts at a definite point in the test model. From that point, the algorithm iterates backward in the state machine to the primary configuration with a guided depth-first search process and creates a corresponding path from the graph. While moving backward, the algorithm collects all situation and keeps them in a consistent set of dataflow information.

```
Test Casecreate TestCase(te : Trace Extension)
{
n = target node of the last transition of te;
TestCasetc = search BackwardsFromNode(n, te);
if(tc is a valid test case)
{
returntc;
}
else {
return null;
}
}
}
Test Casesearch Backwards FromNode(n : Node, te : Trace Extension) {
if(n is initial node and all expressions are satisfied)
{ // valid
return test case that contains the current path information;
}
TestCasetc = null;
if(n has a transition t that is part of te)
{
tc = traverseTransition(t, te);
if(tc != null)
returntc;
}
Else
{
for each incoming transition t of n {
tc = traverse Transition(t, te);
if(tc != null)
returntc;
}
}
return null;
}
```

CONCLUSIONS AND FUTURE WORK;

Software testing is a process of executing software in a controlled way and where regression testing is an expensive but basic maintenance activity performed on modified software to give confidence that changes are accurate and do not badly affects other portions of the software.

Models are an outstanding way to symbolize and understand system behavior, and they give an easy way to renew tests to keep pace with applications that are always changing and developing. Testing an application can be viewed as traversing a path throughout the graph of the model. Graph theory techniques therefore allow us to use the behavioral information stored in models to produce new and useful tests. Tests can be continually changing on the same model. Many different types of traversals can meet different requirements of testers. The traversal techniques are common and can be re-used on different models. Model-based testing is a black-box technique that offers many advantages over traditional testing: Initially, constructing the behavioral models can begin early in the development cycle and Secondly, Modeling exposes ambiguities in the requirement and design of the software. The model embodies behavioral information that can be re-used in future testing, even when the specifications change. Moreover the model is very simple to update than a suite of individual tests. And, most importantly, a model furnishes information that can be attached with graph theory techniques to create many different test scenarios automatically. Testing benefits from the fact that the actual system is brought to execution. Thus, the communication of the actual hardware and the actual software can be evaluated. Testing is applicable at different levels of concept and at different stages of the development. With our approach UML state machines can be used in the quality assurance to provide as a specification for the preferred reactive behavior of the system. It is possible to choose applicable and interesting inputs for a test case and to compute the possible accurate explanation for given inputs. They permit to automatically evaluating test executions which are in common a difficult and time taking task. Applied approximation makes the creation process practical. Here the number of test cases is reduced and they get transition path coverage by testing the boundaries. Moreover, our planning is to contain other diagrams of UML to create test cases. In future, we will try to optimize test cases and how all other UML diagrams can be combined and used to make test cases and also achieve higher coverage.

REFERENCES:

1. AynurAbdurazik and Jeff Offutt, “Generating Test Cases from UML Specifications”, 1999.
2. SupapornKansomkeat and Sanchai Rivepiboon, "Automated- Generating Test Case Using UMLStatechart Diagrams ",SAICSIT 2003.
3. Deng D., et al “Model-based Testing and Maintenance”, Proceedings of the IEEE Sixth (ISMSE’04),pg. 1-8, 2004.
4. M.S.Lund and K. Stolen, “Deriving Tests from UML 2.0 Sequence Diagrams with neg and assert”, AST’06, May 2006.
5. F. Basanieri, A. Bertomated ,E. Marchetti, A. Rinoline, G. Lombardi, and G. Nucerga. “An Automated Test Strategy Based on UML Diagrams”. InProceeding of the Ericsson Rational User Conference, UpplandsVasby Sweden, October 10-11,2001,
6. A.Rountev, S.Kagan and J. Sawin, “Coverage Criteria for testing of objectinteractions in sequence diagrams”, In Fundamental Approaches toSoftware Engineering,Edinburgh,Scotland,2-10 April 2005.
7. Emanuela G, Franciso and Patricia, “Test Case Generation by means ofUML sequence diagrams and Labeled Transition Systems,” IEEE ,pp.1292-1297, 2007.
8. V. Garousi, L. Briand, Y. Labiche. “Control flow analysis of UML 2.0 sequence diagrams”, Technical report.Available athttp://www.sce.carleton.ca/squall/pubs/tech_report/TR_SCE-05-09.pdf.
9. Rountev, A., Volgin, O., Reddoch, M.: Control flow analysis for reverseengineering of sequence diagrams. Technical Report OSU-CISRC-3/04-TR 12, Ohio State University, 2004.
10. G. Booch, J. Rumbaugh, and I. Jacobson, The Unified Modeling Language User Guide.Addison-Wesley, 2001.
11. A. Nayak and D. Samanta, “Automatic Test Data Synthesis using UML Sequence Diagrams”. Journal of Object Technology, Vol. 09, No. 2, pp. 75-104, March-April 2010.