International Journal of Information Technology and Management

Vol. VIII, Issue No. XII, May-2015, ISSN 2249-4510

UNIFIED MODELING LANGUAGE AND MODEL BASED TESTING FOR AUTOMATIC TEST CASE GENERATION

AN INTERNATIONALLY INDEXED PEER REVIEWED & REFEREED JOURNAL

www.ignited.in

International Journal of Information Technology and Management
Vol. VIII, Issue No. XII, May-2015, ISSN 2249-4510

Unified Modeling Language and Model Based Testing For Automatic Test Case Generation

Shiv Kumar[1] Dr. Rajan Anand Malik[2]

[1]JJTU Scholar, Jhunjhunu

[2]Director JEMTEC Greater Noida

Abstract – Model-based testing is a software testing technique in which the test cases are resulting from a model that describes the functional features of the system under test. It makes use of a model to generate tests that includes both offline and online testing.

Keywords: Model-Based Testing, Testing Technique, UML

- - - - - - - - - - - - - - X - - - - - - - - - - - - - -

1. INTRODUCTION

Software Testing is important because we all make mistakes. Some of those mistakes are unimportant, but some of them are expensive or dangerous. We need to check everything and anything we produce because things can always go wrong – humans make mistakes all the time [1].

Model-based testing is a software testing technique in which the test cases are resulting from a model that describes the functional features of the system under test. It makes use of a model to generate tests that includes both offline and online testing.

1.1 Importance of Model-Based Testing:

- Unit testing won't be enough to check the functionalities
- To make sure that the system is behaving in the same sequence of actions.
- Model-based testing technique has been adopted as an included part of the testing process.
- Commercial tools are urbanized to support model-based testing.

1.2 Advantages:

- Higher level of Automation is achieved.
- Exhaustive testing is possible.
- Changes to the model can be easily tested.

Since we assume that our work may have mistaken, hence we all need to check our own work. However some mistakes come from bad assumptions and blind spots, so we might make the same mistakes when we check our own work as we made when we did it. So we may not notice the flaws in what we have done.

UML is not a programming language but tools can be used to generate code in various languages using UML diagrams. UML has a direct relation with object oriented analysis and design. After some standardization UML is become an OMG (Object Management Group) standard.

1.3 Goals of UML:

A picture is worth a thousand words, this absolutely fits while discussing about UML. Object oriented concepts were introduced much earlier than UML. So at that time there were no standard methodologies to organize and consolidate the object oriented development. At that point of time UML came into picture.

There are a number of goals for developing UML but the most important is to define some general purpose modeling language which all modelers can use and also it needs to be made simple to recognize and use.

UML diagrams are not only made for developers but also for business users, common people and anybody interested to recognize the system. The system can be a software or non-software. So it must be clear that UML is not a development method

rather it accompanies with processes to make a successful system [1].

At the conclusion the goal of UML can be defined as a simple modeling mechanism to model all possible practical systems in today's complex environment.

2. REVIEW OF LITERATURE:

Increasing software products quality is generally agreed that manual testing is becoming a bottleneck and is a frequent cause of project delays especially for large programs. Therefore, automatic test case design has become important to ensure the quality of present day large software products [2]. Reducing number of test cases: Generation efficient test cases are the essential passport for simplifying the test work and improving the test efficiency. The test work is inefficient because of the great number of the initial test cases, so some Automation algorithms are needed to optimize the test cases [3].

Covering all system requirements - Automating the test case generation process provides a means to ensure that test cases have been derived in a consistent and objective manner and that all system requirements have been covered [4].

2.1 Test case optimization:

Genetic algorithms [5] represent a class of adaptive search techniques and procedures based on the process of natural genetics and Darwin's principle of the survival of the fittest. Genetic algorithm searching mechanism starts with a set of solution called a population. One solution in the population is called a chromosome. The search proceeds for a number of generations, for each generation the fitter solutions (based on the fitness function) will be selected to form a new population. During the cycle, there are three main operators namely reproduction, crossover and mutation. The cycle will repeat for a number of generations until certain termination criteria are met. In [6] a method of generating test cases for structural testing based on genetic algorithms to cover multiple target paths in one run is presented. First, the problem of generating test cases is formulated as a multi-objective optimization problem in which the number of objectives decreases along with generation of test cases. Then, test cases are generated using genetic algorithms incorporating with domain knowledge. In [7] a Constraint-based Genetic Algorithm technique is used to generate optimized test cases from UML Activity diagram and Collaboration diagrams.

3. A CONCEPTUAL MODEL OF UML [1]:

To recognize conceptual model of UML first we need to clarify *what is a conceptual model?* And *why a conceptual model is at all required?*

- A conceptual model can be defined as a model which is made of concepts and their relationships.
- A conceptual model is the first step before drawing a UML diagram. It helps to recognize the entities in the real world and how they interact with each other.

As UML describes the real time systems it is very important to make a conceptual model and then proceed gradually. Conceptual model of UML can be mastered by learning the following three major elements:

- UML building blocks
- Rules to connect the building blocks
- Common mechanisms of UML

UML is a modeling language used to model software and non-software systems. Although UML is used for non-software systems the emphasis is on modeling object oriented software applications. Most of the UML diagrams discussed so far are used to model different aspects like static, dynamic etc. Now what ever be the aspect the artifacts are nothing but objects.

If we look into class diagram, object diagram, collaboration diagram, interaction diagrams all would basically be designed based on the objects.

So the relation between OO design and UML is very important to recognize. The OO design is transformed into UML diagrams according to the requirement. Before concerned the UML in details the OO concepts should be learned properly. Once the OO analysis and design is done the next step is very easy. The input from the OO analysis and design is the input to the UML diagrams.

3.1 UML Diagrams

➢ **UML Architecture**

Any real world system is used by different users. The users can be developers, testers, business people, analysts and many more. So before designing a system the architecture is made with different perspectives in mind. The most important part is to visualize the system from different viewer's perspective. The better we recognize the better we make the system.

UML plays an important role in defining different perspectives of a system. These perspectives are:

- Design
- Put into practice

- Process
- Deployment and the centre is the **Use Case** view which connects all these four. A **Use case** represents the functionality of the system. So the other perspectives are connected with use case.
- **Design** of a system consists of classes, interfaces and collaboration. UML provides class diagram, object diagram to support this.
- **Put into practice** defines the components assembled together to make a complete physical system. UML component diagram is used to support put into practice perspective.
- **Process** defines the flow of the system. So the same elements as used in *Design* are also used to support this perspective.
- **Deployment** represents the physical nodes of the system that forms the hardware. UML deployment diagram is used to support this perspective.

3.2 UML Building Blocks

As UML describes the real time systems it is very important to make a conceptual model and then proceed gradually. Conceptual model of UML can be mastered by learning the following three major elements:

- UML building blocks
- Rules to connect the building blocks
- Common mechanisms of UML

The building blocks of UML can be defined as:

- Things
- Relationships
- Diagrams

Unified modeling language and model based testing

- UML is a standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems [1].
- UML was created by Object Management Group (OMG) and UML 1.0 specification draft was proposed to the OMG in January 1997.
- OMG is continuously putting effort to make a truly industry standard.
- UML stands for Unified Modeling Language.
- UML is different from the other common programming languages like C++, Java, and COBOL etc.
- UML is a pictorial language used to make software blue prints.

CONCLUSION:

This paper concludes that UML can be described as a general purpose visual modeling language to visualize, specify, construct and document software system. Although UML is generally used to model software systems but it is not limited within this boundary. It is also used to model non software systems as well like process flow in a manufacturing unit etc.

REFERENCES:

1. http://www.tutorialspoint.com/software_testing/software_testing_overview.htm

2. S. K. Swain, D. P. Mohapatra and R. Mall, “Test case generation based on state and activity models”, Journal of Object Technology, vol. 9, no. 5, (2010), pp. 1 – 27.

3. Q. Li and J. Li, Proceedings of the International Symposium on Intelligent Information Systems and Applications, (2009).

4. S. J. Cunning and J. W. Rozenblit, “Test scenario generation from a structured requirements specification”, journal of Intelligent and Robotic Systems, vol. 41, no. 2-3, (2005), pp. 87-112.

5. A. Arcuri and X. Yao, “Search based software testing of object-oriented containers”, Information Sciences, vol. 178, no. 15, (2008) August, pp. 3075-3095.

6. G. Dunwei, Z. Wanqiu and Z. Yan, Chinese Journal of Electronics, vol. 19, no. 2, (2011).

7. B. N. Biswal, S. S. Barpanda and D. P. Mohapatra, International Journal of Computer Applications, vol. 1, Issue 14, (2010).