Testing UML Designs using Class, Sequence and Activity Diagrams

Bhavnesh Kumar  
M. Tech Student  
Department of Computer Science & Engineering  
U.I.E.T, Kurukshetra University, Kurukshetra, Haryana, India

Kulvinder Singh  
Faculty  
Department of Computer Science & Engineering  
U.I.E.T, Kurukshetra University, Kurukshetra, Haryana, India

Abstract

Testing is performed by different types of strategies. Generally testing is performed on code, but if the software can be tested in the earlier phases then most of the errors can be eliminated and can be stopped from propagating to next phase. The proposed work presents a novel design based testing approach that can fix errors in initial phase. To perform design based testing, we need a language that can deal with the design ably i.e. Unified Modeling Language (UML). UML, which supports object-oriented technology, is widely used to describe the analysis and design specifications of software development. UML models are an important source of information for test case design. 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. However UML Sequence diagram describes the way in which different objects interacts with each other, sequence of message passing between different objects. And Class diagram identifies the different classes, its attributes and operations respectively. We propose a method to generate test cases using UML activity diagram. Next, we propose a novel approach to generate test cases from test scenarios using UML activity, sequence and class diagram. First we generate xml from UML diagram. UML consists of different designs that are used to specify the static and dynamic behavior of the software. Proposed system focuses on three diagrams viz. class, sequence and activity diagrams. The class diagrams help us to generate static test cases and the sequence and activity diagrams give dynamic test cases and integrity test cases.

Keywords: UML, Class Diagrams, Activity Diagrams, Sequence Diagrams

I. INTRODUCTION

On behalf of the waterfall software development life cycle (SDLC) model; a software life cycle is consist of five phases which are (I) requirements (II) design (III) implementation (IV) software testing (V) maintenance. Execution of a program is done in software testing phase and the system works with an intention of finding the errors. With the increase in the complexity and size of the software system; there will be a requirement of more and more time and manpower for testing. Testing process is done in three parts: test case generation, test execution and test evaluation. Test case generation is the most difficult in all the three parts. Now we will discuss about UML testing.

A. Unified Modeling Language

UML is a standard visual modeling language which is designed for specifying, visualizing, constructing and documenting of the artifacts of software systems. A number of diagrams for describing particular aspects of software artifacts are provided by UML. These diagrams are classified on the behalf whether they are used for describing structural or behavior aspects of the systems. The diagrams under consideration for this particular task are:

B. Class Diagram

The class diagram represents the static view of an object oriented application. It describes the attributes and operations of a class and also the constraints imposed on these members. It is not only used for visualizing, describing and documenting different aspects of a system but it also can be convert to executable code of the software application.

The class diagrams are most important component used in the modelling of object oriented systems because they are the only UML diagrams which can be directly associated with object oriented languages and hence help in automatic code generation.
Fig. 1: Class Diagram for Banking system

C. Exported XML

```xml
<uml:Class name="ATMLog" xmi.id="EAI_4D98C62C_A194_425f_B661_C33E9B676805" visibility="public" namespace="EAPK_2183C10C_5BBA_4379_BE33_26D779DAFD38" isRoot="false" isLeaf="false" isAbstract="false" isActive="false">
<uml:Classifier.feature>
  <uml:Attribute name="LogMsg" changeable="none" visibility="public" ownerScope="instance" targetScope="instance">
    <uml:Expression/>
    <uml:StructuralFeature.type>
      <uml:Classifier xmi.idref="eaxmiid1"/>
    </uml:StructuralFeature.type>
    <uml:ModelElement.taggedValue>
      <uml:TaggedValue tag="type" value="int"/>
      <uml:TaggedValue tag="containment" value="Not Specified"/>
      <uml:TaggedValue tag="ordered" value="0"/>
      <uml:TaggedValue tag="position" value="0"/>
      <uml:TaggedValue tag="lowerBound" value="1"/>
      <uml:TaggedValue tag="upperBound" value="1"/>
      <uml:TaggedValue tag="duplicates" value="0"/>
      <uml:TaggedValue tag="ea_guid" value="{331E56D5-EE9A-448a-AA8B-360E46381E2D}"/>
      <uml:TaggedValue tag="ea_localid" value="4"/>
      <uml:TaggedValue tag="styleex" value="volatile=0;"/>
    </uml:ModelElement.taggedValue>
  </uml:Attribute>
  <uml:Operation name="addLog" visibility="public" ownerScope="instance" isQuery="false" concurrency="sequential">
  <uml:ModelElement.taggedValue>
    </uml:Operation>
</uml:Classifier.feature>
```

All rights reserved by www.ijirst.org
D. Sequence Diagram
Sequence diagram is one of the common interaction diagrams used in UML, which focuses on the message interchange between a number of object lifelines. Sequence diagram describes an interaction by describing the sequence of messages that are exchanged, and also explains their corresponding occurrence specifications on the lifelines.

![Sequence Diagram]

**Fig. 2:** A Sequence Diagram Showing Various Notations [10]

E. Exported XML

```xml
<uml:collaboration xmi:id="EAI_D_2183C10C_E8BA_4379_BE33_26D779DAFD38_Collaboration" name="Collaborations">
  <uml:namespace ownedelement/>
  <uml:classifierrole name="ATMLog" xmi:id="EAI_D_A81BB782_EB8D_424a_93EE_CBB638883F80" visibility="public" base="EAI_D_11111111_5487_4080_A7F4_41526CB0AA00">
    <uml:collaboration interaction>
      <uml:interaction xmi:id="EAI_D_2183C10C_E8BA_4379_BE33_26D779DAFD38_INT" name="EAI_D_2183C10C_E8BA_4379_BE33_26D779DAFD38_INT">
        <uml:message name="printLogg" xmi:id="EAI_D_4CB09944A BFDE_4dc3_8F6A_C080BA9F9913" sender="EAI_D_B00D787C_8974_48dc_BCFC_9A9E76B2FD87" receiver="EAI_D_A81BB782_EB8D_424a_93EE_CBB638883F80">
          <uml:modelelement.taggedvalue>
            <uml:taggedvalue tag="style" value="1"/>
            <uml:taggedvalue tag="ea_type" value="Sequence"/>
            <uml:taggedvalue tag="direction" value="Source -&gt; Destination"/>
          </uml:modelelement.taggedvalue>
          <uml:taggedvalue tag="seqno" value="8"/>
        </uml:message>
        <uml:message name="ea_sourceName" value="ATMLog"/>
        <uml:message name="ea_targetName" value="ATMLog"/>
        <uml:message name="ea_sourceType" value="Sequence"/>
        <uml:message name="ea_targetType" value="Sequence"/>
      </uml:interaction>
    </uml:collaboration>
  </uml:classifierrole>
</uml:collaboration>
```

All rights reserved by www.ijirst.org
F. Activity Diagram

![Activity Diagram for Banking System](image)

**Fig. 3: Activity diagram for Banking system**

The UML activity diagrams describe the sequential or concurrent control flow of activities. They are used for modeling the dynamic aspects of an object-oriented system. The UML activity diagrams are also used as the models for the driven test case generation.
UML activity diagram portrays all the important steps carried out during the execution of a single use case. Test cases generated from the UML activity diagrams do not represent any information of the system. They only depict the dynamic behavior of system.

G. Exported XML

<pre><code>&lt;UML:ActivityModel xmi.id="EAIID_2183C10C_E8BA_4379_BE33_26D779DAFD38_ActivityModel" context="EAPK_2183C10C_E8BA_4379_BE33_26D779DAFD38" name="ActivityModel" visibility="public">
    &lt;UML:StateMachine.transitions&gt;
        &lt;UML:Transition xmi.id="EAID_3A301AB6_0209_414f_95EE_C0487D3EA00E" visibility="public" source="EAID_0D379B52_E56B_484e_8508_DBE1AEE001B5" target="EAID_9E752270_2309_46c9_8DCA_70B819E2D521">
            &lt;UML:ModelElement.taggedValue&gt;
                &lt;UML:TaggedValue tag="style" value="3"/&gt;
                &lt;UML:TaggedValue tag="ea_type" value="ControlFlow"/&gt;
                &lt;UML:TaggedValue tag="direction" value="Source -&gt; Destination"/&gt;
                &lt;UML:TaggedValue tag="linemode" value="3"/&gt;
                &lt;UML:TaggedValue tag="linecolor" value="-1"/&gt;
                &lt;UML:TaggedValue tag="linewidth" value="0"/&gt;
                &lt;UML:TaggedValue tag="seqno" value="5_1"/&gt;
                &lt;UML:TaggedValue tag="headStyle" value="0"/&gt;
                &lt;UML:TaggedValue tag="lineStyle" value="0"/&gt;
                &lt;UML:TaggedValue tag="ea_localid" value="68"/&gt;
                &lt;UML:TaggedValue tag="ea_sourceName" value="AskForAmount"/&gt;
                &lt;UML:TaggedValue tag="ea_sourceType" value="Activity"/&gt;
                &lt;UML:TaggedValue tag="ea_targetType" value="Synchronization"/&gt;
                &lt;UML:TaggedValue tag="ea_sourceID" value="37"/&gt;
                &lt;UML:TaggedValue tag="ea_targetID" value="45"/&gt;
                &lt;UML:TaggedValue tag="src_visibility" value="Public"/&gt;
                &lt;UML:TaggedValue tag="src_aggregation" value="0"/&gt;
                &lt;UML:TaggedValue tag="dst_style" value="Union=0;Derived=0;AllowDuplicates=0;"/&gt;
                &lt;UML:TaggedValue tag="virtualInheritance" value="0"/&gt;
            &lt;/UML:ModelElement.taggedValue&gt;
        &lt;/UML:Transition&gt;
    &lt;/UML:StateMachine.transitions&gt;
&lt;/UML:ActivityModel&gt;</code></pre></code>

II. RELATED WORK

In 2013 Parampreet Kaur et al. [35] stated several heuristic means to measure the quality of test suites, e.g. fault detection, mutation analysis, or coverage criteria. These means of quality measurement could also be used to decide when to stop testing. This paper was centered upon coverage criteria. There are many different kinds of coverage criteria, e.g. focused on data flow, control flow, transition sequences, or boundary values. The authors presented 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. In 2014 Amitashree et al. [41] focused an approach to generate test-cases from UML sequence diagram for detecting deadlocks during the design phase. This reduced the effort and cost involved to fix deadlocks at a later stage. This work began with design of sequence diagram for the system, then converting it to intermediate graph where deadlock points were marked and then traversed to get test-cases. The test-cases thus generated were suitable for detecting deadlocks. The proposed work mainly consisted of 3 steps model UML sequence diagram, representing the sequence diagram graph and traversing it to generate test-cases that would be able to detect deadlocks. The used example was a generalized model that showed the test could exercise all paths and had the ability to detect deadlocks. S. Shanmuga Priya et al. [3] proposed idea about test path generation using UML sequence diagram. In this authors told that Software testing played a major role in deciding the delivery of the product as well as in the quality of the product. Testing should be performed in the initial phases in SDLC so that most of the errors could be eliminated. That’s why; testing shouldn’t be isolated to a single phase in SDLC. Test case generation was the challenging part in software testing process. The proposed work presented a model based testing approach from which the test paths were automated. Unified Modeling Language (UML) Sequence Diagram is considered for designing and a case study of Medical Consultation System is taken for the proposed work by the authors.

Saru Dhir [5] described an idea about the impact of UML techniques in test case generation. He told in his paper that UML is a standard language which is used in business modeling for specifying, visualizing and constructing the software artifacts. Different UML strategies and techniques were implemented in the past. This paper explained the UML2.0 testing for test case generation. In this paper, he focused on effective use of UML techniques and test-case generation methods.
Maicon B. da Silveira et al. [6] proposed an idea about generation of scripts for performance testing based on UML Models. The cost of software testing process is very high in respect of the other stages of SDLC. MBT is a technique used for automatic generation of testing artifacts. In this paper authors described a case study which shows the procedure how to implement the MBT process. Results generated automatically by a Software Product Line (SPL) in our method.

III. PROBLEM FORMULATED

All major testing techniques perform testing once the coding has been done. But the most important part of SDLC is software design because if the design is not perfect, then the coder would never get the correct platform to write his code on. So, there must be a technique which is able to test the design of an object oriented software. UML being the most widely used standard for software design, we here present a technique to test the design created. The class diagrams only tell about the contents and associations of a class. So they can be used to generate static test cases. But sequence and activity diagrams give us the dynamic behavior of class, so they give us the dynamic test cases. And to check whether sequence and activity diagrams are as per class diagram specifications, we create the integrity test cases.

IV. PROPOSED METHODOLOGY

The software design created in UML can be exported to formats like mdl, csv, xls or xml formats. Once exported, we can use language like java, matlab or C# to study the details programmatically and then generate test cases. In proposed method, we convert the UML design into XML format and then generate the static, dynamic and integrity test cases. This methodology can be further used to check the compatibility of one particular module design with a previous existing module. The steps to perform the implementation can be summarized as follows:

- Create a modeling project in Enterprise Architect 12.
- Create class, sequence and activity diagrams for the model.
- Export this UML design to XML format.
- Create a C# project.
- Include the XML created in this project.
- Parse the XML code using C# language coding to explore the different diagrams and the associations among the diagrams.
- Generate static test cases from class diagrams by analyzing their associations.
- Parse the sequence and activity diagrams and generate the dynamic test cases.
- Alternatively include the xml for sequence and activity diagram for another module and check its compatibility with previously existing design. This is integrity test case generation.

Tools to be used:
- Visual studio 2012.
- Enterprise Architect 12.

A. Types of test cases

- Static test cases:- These Test Cases Test the UML Design class and sequence diagram for their compatibility test. its check if all life lines exists in sequence diagram exist as classes in class Diagram or not.
- Integrity Test Cases for class and sequence Diagram:- These testing test the classes and Sequence Diagram in the UML Design for their Dynamic behavior i.e. this test checks if all messages call destined towards of life line exist in corresponding class in form of function or not if the messages feel to exist in destination class then the integrity test fails so the sequence diagram is not valid.
- Deadlock Testing:- there is another test for dynamic Behavior of a UML Sequence Diagram wherein we check for existence of any circular reference exists then there is always a probability of a Dead lock happening.
- Test Path Generation from Activity Diagram:- This Particular test cases checks the Activity Diagram for its Function Sequence and Create all Possible test Cases from the activity Diagram their test cases verifies the logic and Structure of Activity Diagram from the UML Designer.
- Summary:- In OOP the Direction of Development of entire Project is Decided by the UML Design any shortcoming in the design can lead to total or particular failure of code and because loss of time, money and resources. The above maintain test cases enable us to perform to complete checks of their major UML Diagram that is class, Sequence and Activity.
Testing UML Designs using Class, Sequence and Activity Diagrams

Fig. 3: Flowchart for main menu

Fig. 4: Static test case generation for sequence and class diagram
Fig. 5: Integrity test case generation for class and sequence diagram

Fig. 6: Deadlock test in sequence diagram
The proposed method successfully generates the static, dynamic and integrity test cases for an UML design. The method also checks if any new module which is being added to an existing module in an OO Project, is compatible with existing module or not. Collaboration of XML code of sequence diagram, activity diagram and design based code of class diagram generated through Enterprise Architect, has been used to generate test cases automatically for dynamic behavior. The proposed tool quite successfully checks for test cases via class, activity and sequence diagrams.

The proposed method worked with Class diagrams, activity and Sequence diagrams to produce static, dynamic and integrity test cases. Moreover the proposed method fails to work with any other format except XML format. The future researchers can extend this research to work with other formats which UML can be exported to. Also if the discrepancies can be demonstrated graphically, that would help the system designers to create more robust designs. Further work can be explored to use formal methods to make system suitable for real and large systems. This technique further can be extended to include all types of software artifacts to predict change for regression testing.

ACKNOWLEDGMENT

I owe this work done by me to my guide Mr. Dr. Kulvinder Singh, without whose support and guidance, this work would never have been possible. I also thank my parents and family for their support and patience.
REFERENCES


