Verification and Validation of UML Diagrams using Checklists

Delgado A., Dias A., and Brito e Abreu F.

***Abstract*— There are several methods for the verification and validation of UML modeling. UML is becoming the most used modeling language in the Software Engineering world, so we consider that this is a subject of some importance. The diagrams considered are Use Case, Class, State chart, Sequence and Collaboration diagrams. For each type, we consider a typical defect and show how it can be solved using verification and validation techniques.**

***Index Terms*—Software Inspections, UML Modeling, Verification and Validation.**

I. INTRODUCTION

There are several methods for verification (check that the modeling is correct) and validation (check that the system fills the initial requirements) of UML diagrams that are used to perform detailed examination of the design or program developed by a human. One method is Guided Inspection [1] that uses test cases, to determine if the model is complete, correct and consistent. Guided inspection supplements the checklist with the testing concept of "coverage". Coverage measures determine how much of the product being inspected has been examined. Test cases are selected from the test plan so that, for example, every use case is represented by at least one test case. According to [1], the authors suggest guidelines for the inspection process so the benefits (Objectivity, Traceability, Testability and Coverage) can be achieved. The Object-Oriented Reading Techniques (OORTs) Inspection is a technique for early defect detection in software artifacts. According to the article referred in [2], with this technique severe defects can be found more cost-efficiently. OORTs consist of seven individual reading techniques, where in each technique either two UML diagrams are compared, or a diagram is read against a Requirements Description. The article makes reference to a controlled experiment conducted at Ericsson unit in Norway where these techniques were applied with good results. Another technique is the use of checklists that present a list of verifications to be done. The inspector has a checklist which are based on a set of specific questions that are intended to guide the inspector during inspection. The inspector has to answer those questions while reading the software document. In this paper we consider the checklist technique. There are testing techniques to make the verification in the end of the modeling, when there is a prototype. However, these techniques can have high economic cost as they are tests that occur in the end of the work and involve a lot of change in the case something may be wrong. Checklist-based V&V techniques can be used any time in the project, namely in the beginning which means lower defect removal costs comparing with testing techniques. In table 1 we present a short description of existing UML diagrams.

This paper is structured as follows. In section II we present taxonomy for modeling defect characterization. In section III a set of checklists, each one for a different UML diagram, is proposed. In section IV we present our conclusions.

II. TAXONOMY PRESENTATION

In this paper we consider six types of errors in UML modeling. **Layout Errors** relate to the inadequate way in which diagrams are arranged. **Traceability** denotes incoherence between diagrams that belong to the same system. **Notation** corresponds to an incorrect use of the UML notation. **Semantic** defects relate to an invalid representation of target domain (UoD) concepts. **Documentation** defects produce an inadequate or even missing justification of analysis and design decisions, usually performed through natural language descriptions. **Naming Conventions** defects represent non conformities with the adopted convention which is a very useful asset to ease software evolution tasks.

Name of the Diagram	Short Description
Activity Diagram	Depicts high-level business processes, including data flow, or to model the logic of complex logic within a system.
Class Diagram	Shows a collection of static model elements such as classes and types, their contents, and their relationships
Communication Diagram	Shows instances of classes, their interrelationships, and the message flow between them. Communication diagrams typically focus on the structural organization of objects that send and receive messages.
Component Diagram	Depicts the components that compose an application, system, or enterprise. The components, their interrelationships, interactions, and their public interfaces are depicted.
Composite Structure Diagram	Depicts the internal structure of a classifier (such as a class, component, or use case), including the interaction points of the classifier to other parts of the system.
Deployment Diagram	Shows the execution architecture of systems. This includes nodes, either hardware or software execution environments, as well as the middleware connecting them.
Interaction Overview Diagram	A variant of an activity diagram which overviews the control flow within a system or business process. Each node/activity within the diagram can represent another interaction diagram.
Object Diagram	Depicts objects and their relationships at a point in time, typically a special case of either a class diagram or a communication diagram.
Package Diagram	Shows how model elements are organized into packages as well as the dependencies between packages.
Sequence Diagram	Models the sequential logic, in effect the time ordering of messages between classifiers.
State Machine Diagram	Describes the states an object or interaction may be in, as well as the transitions between states. Formerly referred to as a state diagram, state chart diagram, or a state-transition diagram.
Timing Diagram	Depicts the change in state or condition of a classifier instance or role over time. Typically used to show the change in state of an object over time in response to external events.
Use Case Diagram	Shows use cases, actors, and their interrelationships.

Table 1 – Types of UML diagrams [4]

III. MODELING CHECKLISTS [3]

In table 2 we present a relationship between the diagrams and the errors considered in this paper.

Note: We shall refer Use Case as UC

	Use Cases Diagram	Classes Diagram	State-Machine Diagram	Sequence/Collaboration Diagram
(L)ayout	X	X		X
(T)raceability			X	X
(N)otation	X	X	X	X
(S)emantic	X	X	X	X
(D)ocumentation	X	X		X
Naming (C)onvention	X	X		

Table 2 - UML diagrams vs. defects considered in the paper

1) Use Case Diagrams

a) Unconventional Identifier

(C) – The UC should be named from the actor's point of view and be verb-like. The actor's name should be a singular noun that represents its role in the system.

b) Missing Frontier

(N) – The actors should be placed out of the system and the use cases in the system.

c) Wrong Actor's Definition

(N) – The actors are represented with a stickman.

(S) – The actors must have semantic meaning in the system.

(D) – The actors should be enough to cover all the use cases in the system

d) Wrong UC's Definition

(N) – The UCs are represented with ovals.

(S) – The UCs must have meaning in the system, should not be repeated and must refer to computational activities instead of human activities.

(D) – The UCs should be associated to an actor (exceptions are <<includes>> and <<extends>> relationships).

e) Wrong UC/UC Relationship

(S) – In an <<includes>> relationship the arrow points from the use case that uses to the use case that is used. In an <<extends>> relationship the arrow points from the use case that is extend to the use case that extends. The stereotypes should be applied. These relationships are only used between use cases.

(N) – In an <<extends>> or <<includes>> relationship the arrow is closed and unfilled with a fine, continuous line.

(D) – In the <<extends>> relationships the extension points are placed in the UC that extends and must refer when it is applied.

f) Wrong Actor/UC Relationship

(N) – The line that unites the actor to the use case must be fine and continuous.

g) Bad Diagram Structure

(L), (N) – The UCs must be hierarchically displayed.

2) Classes Diagrams

a) Unconventional Identifier

(C) – The name of the class must be a singular noun beginning with capital letter.

(D), (C) – The name of an abstract class must be in italics.

b) Wrong Class Definition

(C) – The attributes of a class should be singular, common nouns.

(S) – The attributes must represent characteristics of the class. The operations must represent behavior of the class.

(S), (D) – The class must represent a logical concept with a cohesive group of responsibilities.

(L) – The class should not be overloaded nor under loaded.

(D) – When possible the class should be generalized / specialized. The subclass(es) must be compatible with the superclass(es) and descending classes should have attributes, operations and/or associations that distinguishes them from ascendant classes. Verify if class A inherits from class B and vice-versa (circular specialization).

(N) – In inheritance relationships, the arrow is closed and unfilled and points to the superclass.

c) Wrong Relationship between Classes

(D) – The multiplicity of a class must be indicated. The roles in the association must indicate what the class represents in the association.

3) State Chart Diagrams

a) Unknown Class/Attribute

(T) – The diagram should represent the state of the objects of a class and the modeled states should be kept by existing attributes in the class.

b) Wrong State Definition

(S) – There should be one and just one initial state and one and just one final state.

(N) – The initial state is represented with a filled circle and the final state with a filled circle and a larger circumference. A state is represented with a rounded rectangle and a transition is represented with an open arrow that points from the first state to the second state.

c) Wrong Event and Action's Definition

(S) – The transition begins with an event and triggers another event.

(T) – The action should be available in the class's interface.

4) Sequence and Collaboration Diagram

a) Unknown UC/Scenario

(T) – The diagram should refer to an existent use case/scenario/actor.

b) Unknown Actor

(T) – The actor should be related with the use case/scenario that is depicted in the diagram.

c) Actor/Actor interaction

(S) – The actors do not interact.

d) Unknown Class/Operation

(T) – The class/operation must exist in the classes' diagram.

e) Missing Parameters

(T) – If an operation has parameters they should be included in the operation.

f) Missing Return

(D) – When an object sends a message it should receive an answer.

g) Wrong Return

(T) – The return values should be the same specified in the classes diagram.

h) Bad Diagram Structuring

(L) – In sequence diagrams the objects are displaced along the X axis and the messages are timely ordered along the Y axis. In collaboration diagrams is shown the relative arrangement of the objects and the messages are numbered by time of occurrence.

(D) – The first message is always sent by an actor and is received by an interface object and all the messages to and from an actor must always pass by an interface object. The objects in both diagrams are unique. When an interface object has to "make a decision" or there are complex activities to do it should be added a control object.

(N) – The lifeline of an object is represented with a non-continuous line. The time of execution is represented with a narrow rectangle.

IV. CONCLUSION

During our survey we observed that despite the high importance of this subject for good UML modeling, there is not significant work in the area. We found also that some diagrams are neglected in comparison with others. The summary of the conclusions of our work can be found in table 3. We concluded that in Use Cases diagrams the most common errors are Notation errors and the least common are Traceability errors. In Class diagrams the most common defects are Documentation ones and the least common Traceability. In State-Machine diagrams the most common are Traceability and Semantic defects and the least common are Layout and Documentation ones. In Sequence/Collaboration diagrams the most common is Traceability and the least common are Layout, Notation, Semantic and Naming Convention defects. In a more global scale the most common defect is Notation and the least common is Layout. The most defect prone diagram is Use Cases diagrams and the least prone is State-Machine diagrams. We think that the area of Quality for UML Modeling should be more studied since this modeling language is becoming more widely used around the world.

	Use Cases diagrams	Classes diagrams	State-Machine diagrams	Sequence/Collaboration diagrams
Layout	**1**	**1**	**0**	**1**
Traceability	**0**	**0**	**2**	**5**
Notation	**6**	**1**	**1**	**1**
Semantic	**3**	**2**	**2**	**1**
Documentation	**3**	**4**	**0**	**2**
Naming Convention	**1**	**3**	**1**	**1**

Table 3 – Frequency of defects in UML diagrams

V. REFERENCES

[1] MAJOR, Melissa L.; MCGREGOR, John D., *Using Guided Inspection to Validate UML Models*, See http://www2.umassd.edu/SWPI/TechnicalReview/major-sew99paper.pdf

[2] CONRADI, Reidar [et al], *Object-Oriented Reading Techniques for Inspection of UML Models – An Industrial Experiment*, See http://www.idi.ntnu.no/grupper/su/publ/pdf/ecoop03-oort-experiment-final.pdf

[3] UNHELKAR, Bhuvan, *Verification and Validation for Quality of UML 2.0 Models*, John Wiley & Sons, Inc., Hoboken, New Jersey, 2005

[4] *Introduction to the Diagrams of UML 2.0*, See http://www.agilemodeling.com/essays/umlDiagrams.htm