

# A Systematic Procedure for Testing UML Designs

Trung Thanh Dinh Trong

Computer Science Department, Colorado State University

trungdt@cs.colostate.edu

## Abstract

*Dynamic testing of design models, in which behavioral models are executed, can reveal flaws in the design level before they are implemented, and thus reduce the costs and time to market. This paper presents a systematic procedure for testing UML designs and observing the test results. This procedure is being incorporated into a prototype UML testing tool.*

**Keywords:** Design model, testing, UML.

## 1. Introduction

The Unified Modeling Language (UML) [3] is the de facto industry object oriented modeling language standard. UML design models describe systems using structural diagrams (e.g., class diagram) and behavioral diagrams (e.g., interaction diagrams, activity diagrams and state charts). Design constraints can be defined using the Object Constraint Language (OCL).

Current practice in UML design evaluation consists of walkthroughs and inspections. However, these tasks can only be performed manually and are tedious because of the amount of information that a reviewer needs to track. They also lack the rigor of traditional program testing techniques. In the dynamic design testing approach, executable models of behavior are executed with different test inputs. Such an approach is expected to enhance the ability to detect faults in the models.

Andrews et. al [1] describe an approach for design model testing. Two sets of test adequacy criteria are also proposed. Class diagram criteria are based on association-end multiplicity, generalization and class attributes, and are used to determine different object configurations that a test must result in. Interaction diagram criteria include condition coverage, full predicate coverage, all message paths and collection coverage, and help determine different sequences of messages that must be tested.

While the test adequacy criteria can help in generating test inputs and determining when to stop testing, a system-

atic procedure is needed to execute the tests automatically. This paper presents a procedure that describes how a test is performed and how results are observed and accumulated. These results are used to determine failures and report on the existence of faults.

## 2. Test Procedure

Testing is performed on a UML design model consisting of class diagrams and interaction diagrams. OCL statements can be used in the class diagram to describe design constraints and pre- and post-conditions. We make the following assumptions:

1. The diagrams are syntactically correct. This can be ensured by UML drawing tools.
2. Before each test is performed, a minimum initial configuration is created with the help of a prefix. A prefix contains messages that create controller objects. The initial configuration contains a few controller objects which allow objects of all other classes to be created.
3. Executable semantics of the UML interaction diagram are known. For example, one can use Mellor's semantics [2].

Testing is performed with a test input vector of system events using appropriate parameters. The parameters themselves are part of some test case. We restrict the system events to be call messages. Before testing the test input vector, an object diagram is created showing the initial configuration of the system. For each test input element, we execute the interaction diagram in the design model that describes the system's reaction to the event. During the execution, the object diagram is changed accordingly as objects get created or deleted, or their states changed. Faults in the design model are automatically reported by the test procedure during execution or at the end.

The following algorithm describes the execution of the test procedure on each element in the test input vector. The

algorithm is first applied to the initial message of the corresponding interaction diagram. Then it is recursively invoked on each subsequent message in the interaction diagram.

**Algorithm executeMessage( $M, O$ )**

**Inputs:**

$M$ : the current message that is being processed;

$O$ : the object that receives the message.

**Steps:**

1. If there is a condition (optional):
  - (a) Verify that all variables used in the condition are initialized. If not, report a fault.
  - (b) If the condition cannot be evaluated, report to the tester. If the condition is true, proceed to the next step. Otherwise return.
2. Verify if all variables, such as parameters used in the message, are initialized. If not, report a fault.
3. If  $M$  is not a create message, verify the existence of target object  $O$ . If  $O$  does not exist, report a fault.
4. Check the pre-condition. If the value of the pre-condition cannot be evaluated, report to the tester. If the value is false, report a fault.
5. Execute the message using executable UML semantics:
  - (a) If  $M$  is a create message, initialize a new object. The current object diagram is updated accordingly.
  - (b) For each direct sub-message  $M_i$  of  $M$  (e.g., messages with sequence number 2.1, 2.2 in a collaboration diagram are direct sub-messages of the message with sequence number 2):
    - Apply the algorithm executeMessage( $M_i, O_i$ ).  $O_i$  is the target object of message  $M_i$
    - Update the corresponding variable that holds the optional return value from  $M_i$ .
6. Evaluate the return value of  $M$ . If it cannot be evaluated, report to the tester.
7. Check the post-condition of  $M$ . Because not all information can be shown in the interaction diagram, the testing process can use some information from the post-condition to update the configuration. If the post-condition does not match the current object diagram, report to the tester.
8. Evaluate the current object diagram against the class diagram. If there is any inconsistency, report a fault.
9. Return.

A difficulty in testing UML designs is that models are sometimes incomplete. Therefore, the behavior of the system can be unknown (e.g., a value cannot be evaluated). In such cases, the testing process will report to the tester. The tester can use domain knowledge to resolve and continue the test process, or conclude that there is a fault in the design.

After the execution of each interaction diagram, the tester should check the current configuration against the appropriate specification document. For example, if a use case implies that an object should be created, and that object does not appear in the object diagram, the tester may conclude that there is a fault in the design. Since the specification is usually described informally, this step must be performed manually.

Our test procedure can automatically report the following failure types:

1. A message is sent to a non-existent object.
2. A variable is used before being initialized.
3. A message is sent even though its pre-condition is not satisfied.
4. Inconsistencies exist between the class diagram and the object diagram.

### 3. Conclusions and Future Work

We have presented a systematic procedure for testing UML design models. Our approach allows building a testing tool that can verify a UML design model by executing it. In our approach, the incompleteness of the design model needs to be resolved by the tester. Thus, the tester must understand the design and have knowledge about the problem domain. In practice, design models are usually incomplete because of the need for abstraction.

We are developing a prototype tool that implements our testing procedure. We will empirically evaluate the procedure using different design models and seeding known faults to evaluate fault detection effectiveness.

### References

- [1] A. Andrews, R. France, S. Ghosh, and G. Craig. Test Adequacy Criteria for UML Design Models. *Journal of Software Testing, Verification and Reliability*, 13(2):95–127, April–June 2003.
- [2] S. Mellor and M. Balcer. *Executable UML: A Foundation for Model Driven Architecture*. Addison Wesley Professional, 2002.
- [3] The Object Management Group. The Unified Modeling Language. Version 1.4, OMG, formal/2001-09-67, 2001.