Chapter 19
An Improved Model-Based Technique for Generating Test Scenarios from UML Class Diagrams

Oluwatolani Oluwagbemi
Universiti Teknologi Malaysia, Malaysia

Hishammuddin Asmuni
Universiti Teknologi Malaysia, Malaysia

ABSTRACT

The foundation of any software testing process is test scenario generation. This is because it forecasts the expected output of a system under development by extracting the artifacts expressed in any of the Unified Modeling Language (UML) diagrams, which are eventually used as the basis for software testing. Class diagrams are UML structural diagrams that describe a system by displaying its classes, attributes, and the relationships between them. Existing class diagram-based test scenario generation techniques only extract data variables and functions, which leads to incomprehensible or vague test scenarios. Consequently, this chapter aims to develop an improved technique that automatically generates test scenarios by reading, extracting, and interpreting the sets of objects that share attributes, operations, relationships, and semantics in a class diagram. From the performance evaluation, the proposed model-based technique is efficiently able to read, interpret, and generate scenarios from all the descriptive links of a class diagram.

INTRODUCTION

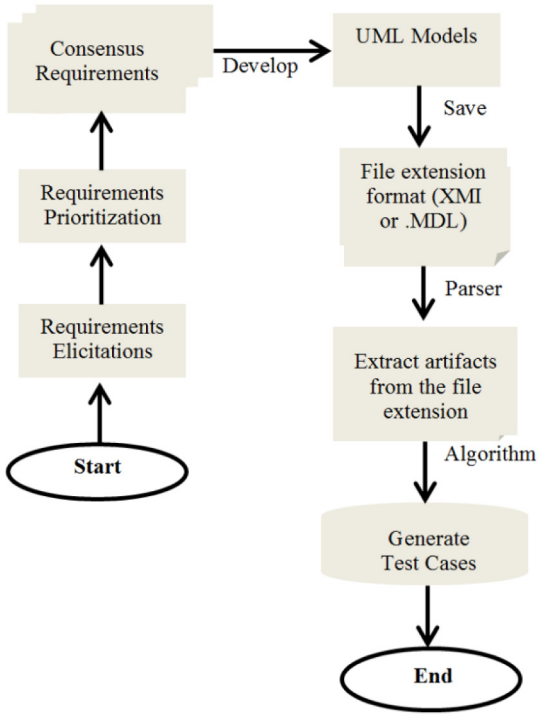
Model-based testing (MBT) is an approach used to assess the quality of software systems based on modeled requirements as captured during the requirements engineering phase of the system development life cycle processes (Prasanna & Chandran, 2009). MBT technique utilizes modeling tools used in representing stakeholder's requirements to extract artifacts and generate test scenarios (Machado & Sampaio, 2010). These modeling tools can be Unified Modeling Language

DOI: 10.4018/978-1-4666-6026-7.ch019

(UML), ArgoUML, Magic Draw or UML Rational Rose among others. Model-based software testing has to do with the creation of test cases from abstract software models which are eventually used to conduct software conformance testing (Sawant & Shah, 2011). Figure 1 depicts the processes involved in model-based software testing.

MBT consists of three basic flows of procedural events described as follows: (i) the modeling tool used in representing stakeholder's requirements (ii) the parser required to extract artifacts from the modeling diagram and (iii) a test case generation algorithm. From literature, most of the techniques for generating test cases in model-based software testing dwell on sequence, activity, state chart, and collaboration diagrams. Class diagram-based test scenario generation techniques are few. The reason may be due to the complexities associated with extracting all the attributes, classes, associations, generalizations, aggregations and compositions in class diagrams so as to generate comprehensive scenarios.

Figure 1. Model-based software testing process



The major activities that take place during model-based testing as shown in Figure 1 are described below:

1. **Model Development:** This phase has to do with the construction of a UML-based diagram that reflects the specified or prioritized requirements using any of the modeling tools. The aim of this phase is to generate a test enabled model that will contain unambiguous artifacts required to generate test scenarios. In this research, the proposed technique was validated using ArgoUML tool because it is open source.
2. **Parser:** Once the modeled diagram is completed, the next task is to save it. UML stores its diagram in an .MDL file extension while ArgoUML stores its diagram in XMI file extensions for example. Therefore, a fundamental task in model-based software testing is the implementation of a parser that has a robust capacity of extracting artifacts from the file extensions of the relevant modeling tool. In this research, a parser was developed and implemented using Java programming language.
3. **Test Scenarios Generation:** These are derived from the parsed artifacts. The parsed artifacts are executed to generate and display test scenarios.

MBT enables testing processes to commence as soon as the requirement specifications and design documents are ready. It also reduces testing time since the testing and development processes can occur concurrently. Therefore, each output of a coding exercise can be compared to the generated test scenarios in order to determine whether the system under development is behaving as expected or not. With MBT, software systems are hardly rejected by stakeholders because each output of the development life cycle can be compared to the generated test scenarios to ensure conformance.