Web Based Software Modeling Exercises in Large-Scale Software Engineering Courses

Birgit Demuth, Daniel Weigel
Technische Universität Dresden
birgit.demuth@tu-dresden.de

Abstract

We present a web based eLearning system to support software modeling exercises in large-scale software engineering courses. Students get the task to create a domain model based on a given textual specification of an application domain. They have to specify the domain model by a UML class diagram and import it into the eLearning system. The system verifies the student's solution and compares it with a set of predetermined sample solutions. This approach is a first step towards an automatic testing of object-oriented models under the conditions of university classes with a large number of students. We discuss it and report about initial experiences in using the eLearning system.

1. Introduction

Software development can only be learned by training. In doing so, students should have fun modeling and writing software. For teachers, however, teaching software modeling and programming is mainly a tedious and time-consuming job. It needs a lot of guidance and feedback to the individual student in the exercises particularly during the training. This induces a resource problem, particularly if a teacher has to supervise and guide a large number of students.

In recent years, several tools for teaching object-oriented techniques have been developed for university environments. Some of them are Praktomat [1], ASB [2], Projectory [3] and EduComponents [4]. All these tools help students and teachers to practice programming through various strategies. To the best of our knowledge, however, there is no tool that supports the automatic testing of object-oriented models. It is clear that the automatic testing of software models is a considerably more difficult, if not impossible task. Modeling represents, to a high degree, a subjective view of the world. Mostly, there is not only one solution for modeling tasks even though teachers try to formulate textual specifications very precisely. But in our university, eLearning scenarios are a hard requirement in teaching hundreds of students. So we have to instruct up to 600 beginners in object-oriented techniques every year. Therefore, we decided to develop a verification tool for an exclusive kind of software models. Because we and our students already gained positive experience in using the eLearning system Praktomat, is used in our eLearning environment for our prototyping purposes. The Praktomat is a web based system that we installed in our department1. It was adapted from the original system presented in [1].

We establish the Praktomat eLearning environment in “early” software engineering (SE) courses. Our approach to SE education is presented in [5]. The main focus in the basic education of software engineers over two semesters is centered on object-oriented methods such as software modeling with UML in analysis (OOA) and design (OOD), implementation and testing by Java, and using design patterns. In a first SE semester, students acquire basic

1 http://praktomat.inf.tu-dresden.de
object-oriented knowledge and skills in lectures and paper and pencil lessons. Subjects in these lessons are among other things small modeling (OOA and OOD) exercises. Within this semester, we additionally offer modeling and Java exercises by the Praktomat. There we assume programming-in-the-small skills that the students gained in previous courses. In the second SE semester students have to apply in software development teams their knowledge in a software project course. This course raises the claim to practice the programming-in-the large in a extensively realistic project [5], [6].

In the following, we first report about specific requirements for tools to support web based software modeling exercises (Sect. 2). In Sect. 3, we briefly present the originated tools. These two tools that work together to test the student's software models against predetermined sample solutions were used in the last winter semester in our SE course. Although it was only a limited experiment in using these tools we gained first experience that we discuss in Sect. 4. Section 5 summarizes our results.

2. Requirements

We consider web based software modeling exercises as addition to our conventional paper and pencil exercises. In classrooms we discuss modeling issues independent of any tool usage. However, we entice students to solve modeling tasks in a self-contained manner as homework. At this point, we have covered a big resource problem to give students feedback to their solutions.

As explained above, our software engineering lectures and exercises for beginners are Java and UML centered. According to the software lifecycle, we distinguish between analysis and design models.

In a first step, we discuss analysis models that are concretely static class diagrams that result, for example, from applying of the CRC (Classes Responsibility Collaboration) card technique. Basically, we understand these analysis models as domain models that are characterized by

- (concrete) classes with
  - attributes specified only by their names
  - methods specified by their names and optionally by a return type and parameters without data types
- is-a relationships (inheritance)
- binary associations/aggregations/compositions that can be adorned by multiplicities, role names and association classes.

From the didactic point of view, it should be noted that we did not like to show the structural differences of the student's model to a sample solution as part of the output. Therefore, we could not reuse algorithms like UMLDiff [7]. Instead we decided to check the student's models in a two-step procedure:

1. Check a student's model for formal requirements whether this model is a well-formed analysis model

2. Comparison of a student's model with equivalent sample solutions and output of helpful comments for the student
A teacher has to create modeling exercises and adequate sample solutions whereas generally multiple solutions belong to one modeling exercise. The first challenge for the teacher is to formulate a precise and comprehensible description of the domain that has to be modeled. The following extract of an example is suggestive of such a specification:

*A city cleaning company that is named employs people and manages customers, containers and vehicles. Their task consists in the operational planning of all resources for the purpose of city cleaning by order of their customers. A vehicle has a type and a license number. An employee has a name, a personnel number (id) and a role in the company. A customer has a name, a customer number, an address and a bank account. Each customer can access connection objects in the city (as places for garbage cans). Each connection object must be exactly attached to one customer. A connection object has a number and is described by the address of the place for the garbage cans. In addition, for each connection object the garbage cans with their type (such as bio or residual waste cans), their volume and the corresponding rate are listed. The clearance of the garbage cans is carried away by a company vehicle and is controlled by an employee. At each clearance, the date, the status (for example, clearance is not possible) and the number of the emptied cans are recorded. In addition to garbage cans, there are further can types ...*

Given the modeling task, the teacher should be able to easily specify a sample solution including the specification of alternative and valid solutions by a dedicated language. Furthermore, the teacher is responsible for the configuration of the behavior of the eLearning system such as the output of adequate comments and the severities of comments or failures. So he must be able to declare comments and severities as part of a sample solution. Fig. 1 represents the outlined use cases both for the student and the teacher. This diagram only considers use cases that refer to the verification of software models and not the other installed eLearning scenarios in the Praktomat.

![Use case diagram for web based software modeling exercises](image-url)
Technical requirements are due to the free availability of UML tools for academic purposes and the chances to represent and handle UML models. Everybody in the UML community knows the great idea behind the interchange of UML models by XMI, but, on the other hand, also the problems with existing XMI adoptions and their interchange between heterogeneous tool environments [8]. According the state of the art, we were looking for tools that support UML 2.x and the adapted import of models by the XMI 2.x format. Furthermore, the verification tools have to be integrated into the Praktomat environment.

3. Tools

According to the in Sect. 2 defined requirements, we decided to develop for each step a tool, the UMLChecker and the UMLTutor. Both work together in the Praktomat environment [9]. As basis implementation for both tools we reuse the EMF based implementation of the UML 2.x metamodel Eclipse UML2\(^2\). Therewith, all UML tools that are based on the EMF UML2.x metamodel are basically supported. We suggest our students draw their models with TopCase\(^3\) or MagicDraw\(^4\), and import the resulting XMI files into the Praktomat. The teacher has to prepare the configuration of the system and the sample solutions by XML.

3.1. UML Checker

The UMLChecker is a framework for the execution of various verifications of UML models that are uploaded by the students. The function of the UMLChecker is to ensure that the UMLTutor gets a well-formed UML analysis model as input. Well-formedness of an analysis model is defined by the requirements listed in Sect. 2. For instance, the student's solution is checked if the model contains banned elements like an interface or static attributes. Each well-formedness check is configured by the \(<\text{checks}>\) element in the umlchecker.xml file:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<umlchecker>
  <checks>
    <check class="de.umlchecker.check.impl.AnalysisCheck"
           enabled="true" />
  </checks>
</umlchecker>
```

The current prototype can easily be extended by further well-formedness checks. The UMLChecker lists information, warnings and errors to inform the student about problems in her/his model.

---

\(^2\) [http://www.eclipse.org/modeling/mdt/](http://www.eclipse.org/modeling/mdt/)

\(^3\) [http://topcased.gforge.enseeiht.fr/](http://topcased.gforge.enseeiht.fr/)

\(^4\) [http://www.magicdraw.com](http://www.magicdraw.com), Community Edition
3.1. UML Tutor

The UML Tutor takes a well-formed analysis model as input and compares it with predefined sample solutions.

The teacher’s language to prepare sample solutions is XML again. A sample solution is a by XML comments annotated UML (class) model. The following extract of the above given modeling example shows the associated snippet of a sample solution.

*A city cleaning company that is named employs people ... An employee has a name, a personnel number (id) and a role in the company.*

One of the correct UML class models is represented in Fig. 2.

![Class diagram snippet of the example](image)

Fig.2: Class diagram snippet of the example

A teacher can describe alternative solutions within a sample solution by UML Tutor specific comments in XML. This is indicated in Fig. 2 for the association end at the class Employee. The comment could look as follows:

```xml
<uml_tutor>
  <compare>
    <name regex="(employees|employee)" />
    <multiplicity>
      <lower regex="(0\|1)" />
    </multiplicity>
    <kind regex="(association|aggregation)" />
  </compare>
</uml_tutor>
```

Basically, the student’s model must be consistent with one of the valid sample solutions to be evaluated as correct. The algorithm to check this tries to match classes, association classes, associations, and inheritance relationships at first. The matching algorithm is based on equality of model element names. To be flexible in modeling we support the specification of equivalent model elements by the specification of regular expressions. Such regular expressions cannot be only specified for names but also for multiplicities and the kind of class relationships. In the example above, possible solutions for the role name, the lower bound of the multiplicity and the kind of association are presented.
After the matching step, the matched elements are compared. The comparison (verification) result is a list of differences including a description. These differences are considered as “events” whereas the following types are distinguished:

- A missing element is detected (for example missingClass).
- An unnecessary element is detected (for example unnecessaryGeneralization).
- An equivalent difference to the sample solution is detected (for example differingName).
- An invalid difference to the sample solution is detected (for example wrongMultiplicity).

A type (error or warning) and a helpful comment for the student are assigned to each event. As the teacher can prepare several sample solutions for one task (especially if he will support structural differences as modeling by delegation or inheritance) there are also several verification results. The student, however, should only get one verification result. In this case, we select as output for the student the verification result for the most similar sample solution (with the lowest number of error/warnings).

4. Issues

In this paper, we presented our approach for web based software modeling exercises. Although we are aware the problem to evaluate automatically models, we experiment with such an approach to teach object-oriented modeling in large-scale software engineering courses.

Before we used the UMLChecker/UMLTutor in classrooms, we successfully validated our prototype by a series of scenarios that cover typical errors in the modeling of class diagrams in the analysis phase [9].

The first practical experience we could gain in the last winter semester. Even though it was only a limited experiment, we saw firsthand how software modeling education can be supported by eLearning. In the coming semesters, we will continue with our studies to apply and enhance our approach. What are the most essential experiences?

4.1. Teaching issues

It is too early to conclusively estimate the didactic value of our approach. There were only a few students who experimented anonymously with the UMLChecker/UMLTutor prototype. We logged eight successful model submissions. Other students still had technical problems with their models (see Sect. 4.2.).

It should be emphasized again that we consider our web based eLearning system only as an additional offer to paper-and-pencil lessons. Overall, the Praktomat cannot substitute face-to-face discussions about modeling issues. An intermediate policy between both basic teaching practices is to provide a reviewing functionality by the eLearning system. Reviewing means that the model could be read and annotated as well as assessed by other students or tutors. Other authors also propose and discuss such practices reviewing program code [1], [2]. For the future, we plan to use the built-in reviewing function of the Praktomat to extend the UMLTutor by reviewing scenarios.
A further plan is to provide an automatic verification of state transition diagrams. However, this project still needs research, because it seems easier to verbally specify modeling tasks for (static) domain models than for (behavioral) state transition models. The crucial point is the verbal specification in such a way that the solution space for valid models is not too big and can be expressed by regular expressions over names of model elements as we, up until now, did it in our sample solution specification. A (in tests) proven trick for such a technique is to predetermine additionally to the verbal specification a class model that should be consistent to the state transition diagram to be modeled.

4.2. Technical issues

It should be noted that we have still to solve some technical problems in our prototypes. Some of them result from existing XMI incompatibilities between the various UML tools. For example, problems occur with the recognition of association classes. Much work is needed to bridge the gap between the UMLTutor and several UML tools that are used by the students. But the high acceptance of the EMF based tools in the Eclipse community promises that the situation can be improved in the near future. A further field is to improve the language for teachers to specify useful comments preparing sample solutions.

5. Summary

In the context of a two-semester software engineering education approach for undergraduates, we presented how we arrange web based software modeling exercises as additional learning offer to paper and pencil lessons, and what tools we developed for this purpose. We limit web based exercises to an exclusive category of exercises, namely small modeling tasks in the analysis phase. We showed that it is, to the best of our knowledge, for the first time possible to automatically test not only Java programs but also software models (in a first step static class models). The need for eLearning scenarios in software modeling resulted from the requirement at our university to efficiently and effectively teach large classes of students in software modeling without compromising quality. We gained promising experiences and will enhance the presented approach.

Acknowledgments

The authors would like to thank Moritz Bartl and Martin Lobe for their efforts and ideas to maintain and extend the Praktomat specifically concerning the online modeling exercises. Their practical work considerably contributed to the acceptance of the Praktomat under the students, and therewith to better learning results in our software engineering course.

References


