Support for Management of Programming Assignments – Automated Grading

M. Paralič* and J. Martončik*
* Department of Computers and Informatics, Technical University of Košice, Slovakia
marek.paralic@tuke.sk, jozef.martoncik@student.tuke.sk

Abstract— In the paper we deal with question, how the practical skills in programming of complex – possibly distributed - systems can be achieved in an effective way. Based on our previous experience with the highly specialized system ShareMe, the generic system GLab is introduced. It offers learning environment that provides not only the description of the laboratory tasks to be implemented in Java, but also covers electronic submission, automatic grading and online access to grading and test results of any Java-based application. In distinction to ShareMe it is an open learning environment that could be used potentially for any software project implemented in Java. We focus on the subsystem called GLabTest that support automated grading and peer-to-peer system for sharing files ShareMe is used for its validation.

Key words: Automated grading, E-learning, Framework, Programming Skills Distributed Systems.

I. INTRODUCTION

The first motivation for designing and implementing learning environment that supports the description of the laboratory tasks, electronic submission, automatic grading as well as online access to grading and test results was simple - increasing number of students. As far as we consider the Distributed programming course (where the learning environment was planned to be applied), in the year 2011 choose the course almost 200 students. Distributed programming course is offered to fifth-semester computer science students. The course consists of a lecture and a laboratory. According to the curriculum, the learning objectives in the laboratory comprise deepening the theoretical concepts taught in the lecture (e.g., various models of distributed systems, communication, concurrency, naming services, distributed objects, remote method invocation, and network security) and applying them in practice. The programming skills that students should gain and improve are focused on the Java language.

Second, even stronger motivation, was the very good experience with the ShareMe system – a peer-to-peer system for sharing files that was designed and implemented at the Vienna University of Technology [1]. It was proposed as complex system containing an assignment, which students can incrementally implement, test, and use. Other parts of the system were automatic grading subsystem and subsystem for publishing of the students results.

Disadvantage of such a system was that grading subsystem was tightly connected to the ShareMe assignment (implemented as collection of shell scripts and Java applications), so it is impossible to apply this successful environment to another types of students assignments.

Therefore we carefully analyzed the ShareMe system, formalized the teaching processes behind and proposed a new generic system that is independent of concrete assignment. It offers the possibility for a teacher to propose own assignment and in a semi-automatic way, based on annotated model solution, generate grading subsystem. Generated grading subsystem is then used for automatic evaluation of students’ solutions.

The rest of the paper is structured as follows. In section two the ShareMe system is briefly described and analyzed. Third section contains identified formal processes that could be abstracted from the ShareMe system’s analysis and represents the basic interaction with the GLab System. Section four introduces the GLabTest subsystem – a generic system that provides the description of the laboratory tasks to be implemented in Java and automatic grading functionality. It is the most important part of the GLab framework that also supports electronic submission, as well as online access to grading and test results. Last section provides results of the system’s validation and concluding remarks.

II. SHAREME

The authors of the ShareMe system at the Distributed Systems Group1 at the Vienna University of Technology faces similar problems with high number of students, limited human and SW/HW resources, as well as set the same goal to ensure that every single student knows how to do network programming. They come up with an up-to-date scenario that not only meets the learning objectives, but also interests the students. They picked up ubiquitous peer-to-peer file sharing as adequate application domain. In a peer-to-peer system, each peer is a computing device that serves as a server and as a client.

The following four subsystems make up the final ShareMe system:
1) the peer-to-peer network infrastructure;
2) the search facility within the peer-to-peer network;
3) a way to secure the communication among peers;
4) the user interface that enables file exchange.

In the following subsection we will focus on the testing system approach, because this part of the system is in the centre of our attention.

1 http://www.infosys.tuwien.ac.at/
A. Testing approach of the ShareMe framework

The ShareMe system contains comprehensive part related to the testing and grading of students solutions. In 4 labs, which have the students to solve, tens of test cases per lab were created and are applied after the student’s submission. The ShareMe testing framework uses for testing of submitted labs the following tools:

- shell scripts - executes some testing tasks and also runs prepared testing programs,
- testing programs - written in Java and used for testing of particular programming tasks,
- text files - used for logging of testing process and saving its results.

Altogether there are more than 200 of executed testing tasks, but most of them have similar implementation. During analysis of particular testing tasks we identify these types:

1. **Invoking after invoking check** - after invoking of some method is checked, if another one was invoked.
2. **Element presence check** - checking, if some programming element is included in specified class or interface.
3. **Method check** - verifying, if method is implemented correctly, where correctness is proved either by returning a value or that no exception is thrown after invoking.
4. **Type check** - checking, if specified type (class or interface) extends or implements another type.
5. **Constructor check** - similar to **Method check**, but the subject of testing here is a constructor.
6. **Program exit check** - check if the running program was finished successfully.
7. **File existence check** - checking, if a file was created and exists during the testing.
8. **Serialization check** - checking by use of deserialization, if an object was serialized and written to a file (i.e. was made persistent) correctly.

III. GLAB FRAMEWORK AND ITS TESTING SUBSYSTEM GLABTEST

Based on the analysis of the ShareMe systems – we identified high level educational process, where a generic system could be applied in order to fulfill the in the introduction identified goals. The process is depicted in the Figure 1.

There are three roles of users that will act in GLab system:
- **administrator** – via web-interface takes care about basic settings, taskManager users and courses,
- **taskManager (Teacher)** - manage taskDevelopers and project in courses,
- **taskDeveloper (Student)** - manage only own project implementation.

In the following section we will focus on the “Model Assignment Annotation and Test Generation” activity, where the GLabTest subsystem is involved.

Our solution expects sample project with solved programming assignment (i.e. model solution) together with specific meta-data – annotations of defined types - that is provided by the teacher. Based on this sample project with meta-data (annotated model solution), particular testing tasks can be specified. Data of these specifications are parsed and used for tests source codes.
generation, which could be later manually modified or customized. The created tests will be then used in the evaluation and grading process of submitted projects.

A. Requirements for the testing subsystem

For the proposed testing subsystem the following main requirements were identified:

Specifying tests for programming tasks - system should be able to help teacher in tests design process. The consequence is therefore the need to provide interface for test specifications. The solution we chose was the utilization of an annotation framework, which is used in model solution for specification of particular testing tasks.

Creating tests - based on obtained data from annotated model solution should be system able to generate test source codes, which could be then (if needed) manually customized by a teacher.

Acceptance of submitted projects - before the evaluation phase of the submission itself is processed, there is need for a pre-phase check phase. It includes check on completeness, not allowed string in source codes presence and of course the check, if successful build is possible.

Testing and grading of submitted projects - in last phase system should verify correctness of project's implementation by its evaluation based on generated tests. Each test case will be evaluated to a value of points from predefined range and the results will be stored in a persistent way.

IV. TOWARDS AUTOMATIC GRADING WITH GLABTEST

A. Specifying tests

At first we designed annotation framework, which is used in tests specification process. It includes seven testing task types (see Table I), which were identified during the ShareMe analysis phase.

As far as the assignments could be of different size and in time they can potentially take whole semester, we decided to break the assignment in pieces and defined the corresponding hierarchy. The introduction of hierarchy in the evaluated assignment has also other advantage - possibility to test each part separately, what in end effect causes that some incorrect part could not influence actually the tested one.

Therefore we defined following hierarchy, which is also included in the annotation framework:

- Project - root level, it contains all others entities.
- Lab - logical unit that divides project to several smaller parts.
- Testing task group - it groups testing tasks, which are related to one component or class.

Designed annotation framework also includes special tools which could be used in testing process. These tools are:

- Provide - it means providing files from sample project, which can be used for variety reasons, e.g. providing correct implementation of class, that isn't directly tested, but is needed by tested class.
- Inject - this tool provides injecting of created objects, what is useful in situations when some method creates object which isn't its return value and we want to check it.
- Edit Method - using this, arbitrary method of tested project could be replaced or appended by some code, what is desirable for example by providing correct implementation, checking if was invoked, logging etc.

B. Creating tests

Second step is to generate tests, based on data provided by annotations. For this purpose we utilized the following technologies:

- Annotation processors - it serves for processing of annotations from our annotation framework. We used it for reading specified data and creating model of tests from them. In JDK 1.6.0 and more they could be activated through javac compiler [2].
- Apache Velocity – text file generation software tool, which is developed as open-source project of Apache foundation. In our system is Velocity used for generating test source codes [3].

As we already mentioned, automatically generated tests – i.e. the corresponding source codes in Java – are not necessary complete. It is very difficult to generate test exactly according to teacher's view, because then a lot of specifications would be needed. This would cause increasing complexity of annotations and that would decrease the usability of whole system. Thus we chose compromise in terms of what will be generated and what manually added, modified or customized.

C. Acceptance of submitted projects

Before testing of submitted projects we need to verify, if they match requirements:

- completeness of submitted project - checking if all required files are presented,
- not allowed string occurrence - checking if source codes doesn't contain some code that could affect results of testing,
- build test - before we can run tests, binary version of project is needed, so the possibility for

<table>
<thead>
<tr>
<th>Table I. Testing task types and data used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invoking after invoking check</td>
</tr>
<tr>
<td>Element presence check</td>
</tr>
<tr>
<td>Method check</td>
</tr>
<tr>
<td>Type check</td>
</tr>
<tr>
<td>Constructor check</td>
</tr>
<tr>
<td>File existence check</td>
</tr>
<tr>
<td>Serialization check</td>
</tr>
</tbody>
</table>
creating a successful build of the submitted application is applied.

For this purpose we used beside standard Java compiler also Apache IO Commons that are software tools for manipulating with files, developed as open-source project by Apache foundation [4].

D. Testing and grading of submitted projects

When tests are prepared and submitted projects accepted, then testing and grading process can be started.

For testing purpose we decided to use AspectJ - aspect-oriented extension of Java, which helped us to implement special tools Provide, Inject and Edit method, which were already mentioned [5].

V. ARCHITECTURE OF THE GLABTEST AND ITS IMPLEMENTATION

Architecture of designed system consists of:
- component GLabTestMain - it executes main functions of GLabTest system
- Annotation framework - allows to specify test data using it in sample project
- Annotation processors - collect test data from annotations and transform it to test model
- Test Model - set of data objects which holds information about tests structure, particular testing tasks and special tools to use during testing
- component TestGenerator - generates test source codes based on obtained test model and creates test result storage
- component TestRunner - used generated tests for testing of submitted projects
- component TestExecutor - executes particular tests and manages saving its results and grading.

The high level process that represents the utilization of the GLabTest subsystem is depicted at the Figure 2.

Implementation of the proposed GLabTest system was verified by testing of programming assignment of the original peer-to-peer system for sharing files – ShareMe. We took 20 student submissions of three labs (lab1, lab2 and lab3) and run 37 testing tasks on each of them. These tests included all identified test types that are supported by the annotation framework and are listed in Table 1.

Test results were compared with results provided by original ShareMe testing module. From the overall number of 740 testing tasks, 717 of them finished with the same result. The rest – that represent 3,1% of all executed tests - failed because of little different implementation of some tests, what was obviously traceable. Based on these we can state that the introduced system was designed and implemented correctly.

VI. CONCLUSION AND FUTURE WORK

According to [1] - taking into account the growing number of students and the limited resources at the universities, personal support of each student is no longer possible. Nevertheless, each student has to receive adequate experience with the topics and technologies related to programming courses.

According to [6] there exists no significant difference between Web-based and classroom learning environments as long as 1) the training material is of high quality and 2) the feedback is timely and accurate. The submission and the grading of laboratory assignments should be automated in order the students get immediately the feedback.

In this paper we describe a solution that offers such an environment and enables to support almost arbitrary programming assignment. The only preconditions are Java as a programming language and annotated sample solution provided by the teacher.

In the near future, the prototype implementation of the GLab system that supports also electronic submissions, as well as online access to grading and test results, will be extensively tested in Distributed Programming course.

Figure 2. Process-oriented architecture of the GLabTest subsystem
GLab system is going also to be extended by “Originality testing subsystem” - a component introduced by IT4KT - Information technology for knowledge transfer research project, in order to consider the plagiarism issue [7].

ACKNOWLEDGMENT

This work is the result of the project implementation: IT4KT - Information technology for knowledge transfer (ITMS project code: 26220220123) supported by the Research & Development Operational Program funded by the ERDF.

REFERENCES


