Teaching Computer Programming in Online Courses
How Unit Tests Allow for Automated Feedback and Grading

Edgar Seemann
Furtwangen University, Villingen-Schwenningen, Germany

Keywords: Online Courses, Computer Programming, Assessment, Grading, Teaching, Tutoring, Unit Testing.

Abstract: Online courses raise many new challenges. It is particularly difficult to teach subjects, which focus on technical principles and require students to practice. In order to motivate and support students we need to provide assistance and feedback. When the number of students in online courses increases to several thousand participants this assistance and feedback cannot be handled by the teaching staff alone. In this paper we propose a system, which allows to automatically validate programming exercises at a fine-grained level using unit tests. Thus, students get immediate feedback, which helps them understanding the encountered problems. The proposed system offers a wide range of possible exercise types for programming exercises. These range from exercises where students need to provide only code snippets to exercises including complex algorithms. Moreover, the system allows teachers to grade student exercises automatically. Unlike common grading tools for programming exercises, it can deal with partial solutions and avoids an all-or-nothing style grading.

1 INTRODUCTION

Basic programming is part of the syllabus of any engineering discipline. In fact, programming skills are required for many tasks ranging from automation to data analysis. There is a common consensus, that programming skills cannot be learned from books or lectures alone (Milne and Rowe, 2002). Consequently, programming courses are often accompanied with laboratory classes where students practice programming in front of a computer. Many courses also require students to hand in results of programming projects. Usually teaching assistants supervise students during these laboratory classes and projects.

For online courses we have to cope with new challenges in programming education. Firstly, direct interaction and assistance is more difficult. Online courses mostly use forums to answer student questions. This communication is tedious and since answers are often delayed by hours or days inhibit the learning process.

Even though desktop sharing software would allow for real-time assistance, we are not aware of any educational institution using this technology to a larger extent. In particular since online courses are usually designed for a large student audience, where individual assistance would be too time-consuming for the teaching staff.

When dealing with hundreds or even thousands of students it is also difficult to grade exercise sheets. Revising such an amount of source code manually is no longer feasible. Online universities like Udacity (Thrun et al., 2012) or Coursera (Ng and Koller, 2012) try to remedy this situation by using simple quizzes in their courses. These quizzes are often based on multiple choice questions and are graded automatically. Thus, allowing immediate feedback to the students. For some special courses e.g. (Thrun, 2013) customized tools are used to allow students to hand-in small computer programs. Currently there are, to our knowledge, no general concepts or available frameworks to deal with programming exercises of varying complexity, i.e. ranging from simple code snippets, class and interface implementations to medium or large programming projects.

In this paper we propose a software tool to evaluate different types of programming exercises in online courses. We leverage the capabilities of unit testing frameworks to analyze source code snippets or whole computer programs. Our tool allows teachers to easily create custom electronic exercise sheets which may be automatically analyzed and graded. Students receive individual feedback on their implementations via a web-based user interface. Consequently, they are able to evaluate their own learning process and success.

Seemann E..
Teaching Computer Programming in Online Courses - How Unit Tests Allow for Automated Feedback and Grading.
DOI: 10.5220/0004939304210426
In *Proceedings of the 6th International Conference on Computer Supported Education* (CSEDU-2014), pages 421-426
ISBN: 978-989-758-020-8

2 RELATED WORK

For the purpose of this paper, we divide educational software supporting programming education into two categories. The first category are intelligent tutoring systems tailored to support the understanding of fundamental programming concepts. The other category are automated grading tools which are also often used in the context of programming contests.

Intelligent Tutoring Systems. Tutoring tools for programming education date back to the early 1980's (Johnson and Soloway, 1984; Soloway, 1986). The goal of a tutoring system is to analyse student responses and gather information about the learning process (Brusilovsky, 1995; Corbett and Anderson, 2008). Thus, they are able to assist students, point out weaknesses and strength and suggest further study material.

Tutoring systems typically present learning units in small cognitive accessible exercises or dialogs (Lane and VanLehn, 2004). They may also coach students with hints, tips and additional information throughout the study process (Lane and VanLehn, 2003). This step-by-step process is not always appropriate. Teachers also want students to engage in more complex problem solving tasks without or with less guidance. These types of exercises often cannot be properly covered with tutoring systems.

The progress in research on intelligent tutoring systems has also been accompanied by the evolution of new programming concepts and languages. Particularly, the rise of object oriented programming languages has changed how programming is taught and how tutoring systems are implemented (Sykes and Franek, 2003). While object orientation is certainly the main focus of today's programming education, there are tutors for other popular concepts as e.g. functional programming (Xu and Sarrafzadeh, 2004). Modern tutoring systems are often web-based (Butz et al., 2004) with a central server analyzing source code, building student models and offering help and assistance to students.

Automated Grading Systems. The development of grading tools happened along two different paths. One motivation for their use were large scale programming contests, where students compete in solving algorithmic problems (Leal and Silva, 2003). Usually, problems are designed to require complex program logic, but to produce simple output (e.g. find the shortest route to escape a maze). Some contest software also evaluates source code and program metrics, e.g. number of lines of the implementation or program speed (Leal and Silva, 2008).

In recent years, grading tools have also become more popular as learning support tools, allowing teachers to spend more time working with students and less time grading course assignments (Tiantian et al., 2009; Hukk et al., 2011; Foxley et al., 2001).

Grading tools usually evaluate the correctness of a complete computer program. That is, a complex exercise is either correct or incorrect. For students this is often frustrating. A tiny error and their almost working solution is not accepted. When using such a system at our university we observed that it works well for the top students in a class. Weaker students, however, quickly loose their motivation and do worse than with conventional assignment sheets.

3 ASSESSMENT STRATEGIES FOR ONLINE COURSES

For online courses, we need elements from both tutoring systems and automated grading tools. That is, we want to support and guide students when new concepts are introduced. On the other hand, we want them to solve more complex exercises and evaluate these automatically.

In this paper, we propose a first step in this direction. Our system makes it possible to define exercises of varying complexity. We allow teachers to define exercises with step-by-step instructions similar to quizzes. But we also allow them to define more comprehensive exercises where students may practice algorithmic problem solving skills.

In order to accomplish this goal, we analyze the source code using a unit testing framework. With unit tests, we can test both program parts (e.g. a single method or function) or the program as a whole. Unit tests are on the one hand a familiar tool to any programmer and thus to any computer science teacher. On the other hand, they allow teachers lots of flexibility when designing their exercises.

The idea is to define many unit tests for a single exercise and test program parts instead of the overall functionality. As a consequence, we can return a more fine-grained feedback. Not only is this feedback helpful for students when solving the exercises, unit test also allow to attribute points to the individual parts of an exercise. We are convinced that this leads to a more transparent and fair grading result.

In the following sections we will show two small sample exercises as they have been implemented in our system.

3.1 Simple Exercise

A simple exercise is often used as part of a step-by-step learning process. It requires students to complete only a very small task or a subpart of a larger task. For programming exercises students are often required to fill in a small code snippet, e.g. a specific command.

In the example presented in Figure 1, the student has to insert a snippet for a mathematic operation. We will discuss the validation process for such a code snippet in section 4. Students receive feedback in the form of points, when the code is correct and error messages and hints are displayed if the snippet does not properly solve the exercise.

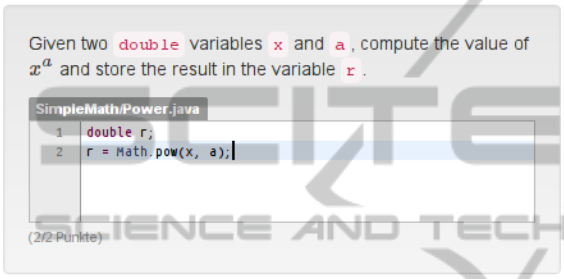

Figure 1: A simple exercise requiring students to enter a small code snippet.

3.2 Complex Exercise

For more complex problems, students typically need to fulfill multiple requirements. Even if the solution is a single class, method or computer program, we want to check for these requirements separately.

In Figure 2 a more elaborate exercise is shown. In this specific example a solution has to satisfy three requirements. It needs to implement a correct method signature with appropriate parameters. The method needs to return a correct return value and finally it has to produce a specific console output by extracting sentences from a text.

In order to check these requirements three unit tests have been implemented. Figure 2 displays a student solution, where only one requirement is met. For the other requirements meaningful error messages and hints are displayed. Thus, we avoid the all-or-nothing style grading and help students to build their solution incrementally.

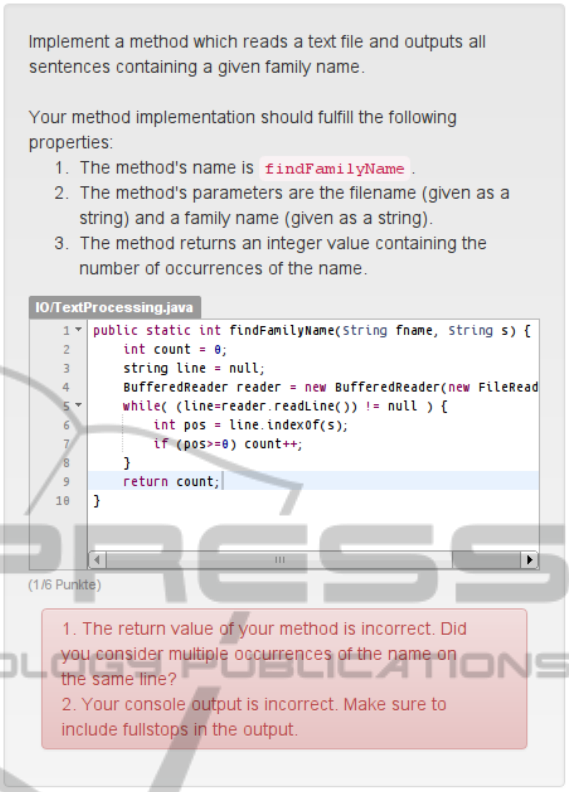

Figure 2: A complex exercise requiring students to implement an algorithm with a specified interface.

4 IMPLEMENTATION AND EXERCISE TYPES

Most grading tools for programming courses only consider the final output of a computer program. This behaviour has serious disadvantages. Checking of console output is susceptible to small typos or differing whitespace characters (e.g. spaces, line breaks or tabulators). These little mistakes can be a source of frustration for students and we would like to avoid penalties for such errors.

Unit tests allow to check programs at a much more detailed level. In fact, we can check individual parts of the program in a type-safe manner. Thus, it is possible to define a much wider range of possible exercises. In the following we systematically present the most common types of exercises and the corresponding solutions.

Code Snippet. A programming exercise may require students to provide a small code snippet (see e.g. Figure 1).

In order to test such a code snippet with a unit test, it needs to be inserted into a surrounding main program. In most cases, this surrounding program defines a method, which returns a value that has been computed in the code snippet. Consequently, we check the return value and return type of this function.

For the example in Figure 1 a main program could

look like this:

```
class Power {
  static double pow(double x, double a) {
    // CODE SNIPPET IS INSERTED HERE
    return r;
  }
}
```

Note that it would not be feasible to use simple string matching to compare a snippet to a pre-defined solution. There are many ways to rewrite code in an equivalent manner. E.g. the shorter expression `double r = Math.pow(x,a);` would work as a solution, too.

Method/Function. Exercises where students need to implement a specific method are naturally suited for unit tests. Again the student's implementation is inserted into a surrounding main program and checked via an appropriate test.

A unit test checking for a method with a computed return value could be implemented in the following manner (compare Figure 2).

```
class TextProcessingTest {
  @Test
  public void checkReturnValue() {
    int c1, c2;
    c1 = Sol.findFamilyName("f1.txt","Hill");
    c2 = Ref.findFamilyName("f1.txt","Hill");
    assertEquals(c1, c2);
    c1 = Sol.findFamilyName("f2.txt","Lee");
    c2 = Ref.findFamilyName("f2.txt","Lee");
    assertEquals(c1, c2);
  }
}
```

For the above example Java along with the JUnit testing framework has been used. For other languages and frameworks this would work in a similar way. Here we check the student solution inserted in the class `Sol` with the reference implementation in the class `Ref`. We compare the results for two different test cases.

Most algorithmic exercises can be tested using tests for one or multiple methods. Also note, that unit tests allow type-safe testing. Thus, we can also process and compare structured data types (i.e. objects) in parameters and return types.

Class with Interface When learning object oriented programming, students need to implement their own classes and data types. This includes the definition of appropriate member variables, constructors and methods.

Often an implementation has to conform to a specific interface, which then may be tested with the help of unit tests. The unit tests create instances of the defined classes and check the public member variables and methods.

While testing the interface of a class is mostly sufficient, some programming languages even allow us to inspect the classes and objects. This inspection is called type introspection or reflection (Forman and Forman, 2005) and it allows us to analyze a program at a very fine-grained level. Using this technology we can easily check the types of private member variables and signatures of class and member methods.

In the example below, we depict a unit test, which compares the return type of each implemented interface method with a reference type stored in a predefined array `returnTypes`. If the types do not match, the test fails and an error message is returned:

```
class PlayerInterfaceTest {
  @Test
  public void checkReturnTypes() {
    Method[] methods = \
      Player.class.getDeclaredMethods();
    for(Method m : methods) {
      Type t = m.getReturnType();
      if (t.equals(returnTypes[m.getName()]))
        fail("Return type of method " \
            +m.getName()+" is wrong");
    }
  }
}
```

Console Output. Finally, we can use unit tests to compare console output. Console output may be used to compare simple programs e.g. a Hello World program or the output of complex algorithms.

One way to do this with unit test, is to redirect the standard output to a string and then do a string comparison. For Java e.g. output redirection can be achieved by providing a new `PrintStream` object:

```
System.setOut(printStreamObject);
```

As pointed out earlier, comparing of console output is not type-safe and susceptible to typos and whitespace errors. This type of checking should therefore only be used for exercises which are designed to practice how to do console output. Other programs should be tested in a type-safe way using unit tests for methods and classes (see above).

5 EXERCISE WORK-FLOW

A practice and learning tool for students should be easy to use. The heavy-lifting of our implementation is therefore done on the server side and is transparent to the user.

The work-flow of an exercise is split into 4 parts:

1. Write and test in an IDE
2. Submit via web-based UI
3. Compile and validate on server
4. Feedback

Write and Test in IDE Students write their implementations in an integrated development environment (IDE). On the hand, we want students to learn how to use IDEs and their convenience features (e.g. code completion). On the other hand, this allows students to compile and run their code on their own test data before submitting it.

Submission. In our implementation exercise sheets are presented in a web-based user interface. In this same user interface submission fields are integrated (see Figure 1 and Figure 2). To support syntax highlighting of source code, we use the ACE Javascript editor [1] allowing students to have a convenient overview over the exercises and their respective responses.

Compile and Validate. Students results are received on a central validation server. The server preprocesses the input (e.g. inserting of code snippet in a main program) and handles the compilation. Compilation errors are reported back to the user interface via AJAX.

Once the compilation succeeds the code is checked via unit tests. For security reasons the compilation and testing is done in a sandbox. Additionally, the server makes sure that no endless loops occur in the student code. This is accomplished by a timeout setting defined in the unit tests. In JUnit this is achieved in the following way:

```
1 @Test(timeout=1000)
2 public void testMethod() {
3   // some test code
4 }
```

Here, the test fails, when its execution takes longer than 1000 milliseconds.

Feedback. As unit tests allow a more fine-grained analysis of the submitted code, we can also provide more detailed feedback to the students. In fact, meaningful feedback is particularly important for online courses, since asking questions in a forum and receiving an answer usually takes a long time.

The proposed system provides feedback to every defined unit test and of course run-time exceptions. This feedback is displayed along with the achieved points directly below the input in the web-based user interface. A teacher can customize this feedback and add additional assistance and hints (see Figure 2 and Section 6).

[1] http://ace.c9.io

6 CONTENT GENERATION

For teachers it has to be easy to create custom content. Interactive exercises therefore may be defined in a simple text format inspired by Markdown (Gruber, 2004). Input fields with their properties are specified with XML tags.

The example in Figure 1 may be created with the following text:

```
??? Math functions
Given two *double* variables *x* and *a*,
compute the value of $x^a$ and store the
result in the variable *r*.
<field:java file="Power.java">
  <test="CheckValueOfR" points="2"
   fail="The value of 'r' is incorrect.
   Make sure that the variable 'a'
   is the exponent.">
</field>  .
```

The character sequence `???` starts a new exercise with the title `Math functions`. The stars highlight the enclosed terms. The more structured XML part specifies which unit test to use and which feedback to return when the unit test fails.

From the above text format an HTML page for the web-based user interface is generated. Additionally, the unit test needs to be implemented e.g. as shown in Section 4.

7 CONCLUSION

In this paper we have presented an interactive learning and grading tool for online courses. The tool lets teachers create programming exercises, which are automatically validated. Students on the other hand get immediate feedback to their solutions and can therefore incrementally improve their abilities.

The proposed system implements a general concept on how to handle programming exercises of varying complexity. Our system allows teachers to use a much wider range of exercises and checks as it is possible with existing tutoring and grading tools. It allows to validate both small code snippets and complex algorithms via unit tests. These unit tests enable a more fine-grained analysis of the source code and a more detailed feedback.

We believe that this improved validation helps student to learn programming in a more efficient way compared to existing online courses. Moreover, the developed tools allow teachers to automatically grade large amounts of student submissions without having to rephrase exercises in the form of multiple choice questions.

REFERENCES

Brusilovsky, P. (1995). Intelligent learning environments for programming. In *Proceedings 7th World Conference on Artificial Intelligence in Education*, pages 1–8.

Butz, C., Hua, S., and Maguire, R. (2004). A web-based intelligent tutoring system for computer programming. In *IEEE/WIC/ACM Conference on Web Intelligence (WI04)*, pages 159–165.

Corbett, A. and Anderson, J. (2008). Student modeling and mastery learning in a computer-based programming tutor. In *Department of Psychology, Carnegie Mellon University*. http://repository.cmu.edu/psychology/18.

Forman, I. and Forman, N. (2005). Java reflection in action. ISBN 1-932394-18-4.

Foxley, E., Higgins, C., Hegazy, T., Symeonidis, P., and Tsintsifas, A. (2001). The coursemaster cba system: Improvements over ceilidh. In *Fifth International Computer Assisted Assessment Conference*.

Gruber, J. (2004). *Markdown text-to-HTML conversion tool*. http://daringfireball.net/.

Hukk, M., Powell, D., and Klein, E. (2011). Infandango: Automated grading for student programming. In *16th Annual Joint Conference on Innovation and Technology in Computer Science Education*.

Johnson, W. and Soloway, E. (1984). Proust: Knowledge-based program understanding. In *7th international Conference on Software Engineering*, pages 369 – 380.

Lane, H. and VanLehn, K. (2003). Coached program planning: Dialogue-based support for novice program design. In *Proceedings of the Thirty-Fourth Technical Symposium on Computer Science Education (SIGCSE)*, pages 148–152.

Lane, H. and VanLehn, K. (2004). A dialogue-based tutoring system for beginning programming. In *Proceedings of the Seventeenth International Florida Artificial Intelligence Research Society Conference (FLAIRS)*, pages 449–454.

Leal, J. and Silva, F. (2003). Mooshak: a web-based multi-site programming contest system. In *Software: Practice and Experience*, volume 33, pages 567–581.

Leal, J. and Silva, F. (2008). Using mooshak as a competitive learning tool. In *Competitive Learning Institute Symposium*.

Milne, I. and Rowe, G. (2002). Difficulties in learning and teaching programmingviews of students and tutors. In *Education and Information technologies*.

Ng, A. and Koller, D. (2012). *Coursera — Take the World's Best Courses, Online, For Free.* https://www.coursera.org.

Soloway, E. (1986). Learning to program = learning to construct mechanisms and explanations. In *Communications of the ACM archive*, volume 20 Issue 9, pages 850 – 858.

Sykes, E. and Franek, F. (2003). A prototype for an intelligent tutoring system for students learning to program in java. In *IASTED International Conference on Computers and Advanced Technology in Education*, pages 78–83.

Thrun, S. (2013). Artificial intelligence for robotics. In *Udacity*. https://www.udacity.com/course/cs373.

Thrun, S., Sokolsky, M., and Stavens, D. (2012). *Udacity — Learn. Think. Do.* https://www.udacity.com.

Tiantian, W., Xiaohong, S., Peijun, M., Yuying, W., and Kuanquan, W. (2009). Autolep: An automated learning and examination system for programming and its application in programming course. In *First International Workshop on Education Technology and Computer Science*, pages 43–46.

Xu, L. and Sarrafzadeh, A. (2004). Haskell-tutor: An intelligent tutoring system for haskell programming language. In *Institute of Information and Mathematical Sciences Postgraduate Conference*.