Research in Learning Technology
Vol. 25, 2017

ASSOCIATION FOR LEARNING TECHNOLOGY ALT

ORIGINAL RESEARCH ARTICLE

Some aspects of grading Java code submissions in MOOCs

Sándor Király[a]*, Károly Nehéz[b], Olivér Hornyák[b]

[a]*Department of Information Technology, Eszterházy Károly University, Eger, Hungary;*
[b]*Department of Information Engineering, University of Miskolc, Miskolc, Hungary*

(Received 7 November 2016; final version received 30 March 2017)

Recently, massive open online courses (MOOCs) have been offering a new online approach in the field of distance learning and online education. A typical MOOC course consists of video lectures, reading material and easily accessible tests for students. For a computer programming course, it is important to provide interactive, dynamic, online coding exercises and more complex programming assignments for learners. It is expedient for the students to receive prompt feedback on their coding submissions. Although MOOC automated programme evaluation subsystem is capable of assessing source programme files that are in learning management systems, in MOOC systems there is a grader that is responsible for evaluating students' assignments with the result that course staff would be required to assess thousands of programmes submitted by the participants of the course without the benefit of an automatic grader. This paper presents a new concept for grading programming submissions of students and improved techniques based on the Java unit testing framework that enables automatic grading of code chunks. Some examples are also given such as the creation of unique exercises by dynamically generating the parameters of the assignment in a MOOC programming course combined with the kind of coding style recognition to teach coding standards.

Keywords: education; massive open online courses; external grader; open edX; computer programming

Introduction

A massive open online course (MOOC) is an online teaching programme that provides unlimited participation for traditional course materials via the web. MOOCs were first introduced in 2008 and emerged as a popular mode of learning in 2012. MOOCs also provide videos, interactive forums to support knowledge sharing, teamwork and community interactions among students, teachers and teaching assistants.

Learning to programme requires more than just watching video tutorials, reading handouts or filling in tests. It requires students to gain practical skills (Robins, Rountree, and Rountree 2003). Consequently, computer programming instruction in an online environment must provide coding tasks and assessable practical programming assignments with prompt feedback (Vihavainen, Luukkainen, and Kurhila 2012).

When developing an e-learning environment, it is particularly important to apply good exercises from real life to increase both students' satisfaction and engagement (Joo, Joung, and Kim 2013). Good assignments help students to consolidate

*Corresponding author: Email: kiraly.sandor@uni-eszterhazy.hu

Citation: Research in Learning Technology 2017, **25:** 1945 - http://dx.doi.org/10.25304/rlt.v25.1945

their knowledge. Assignments should start from the easiest things and move towards the more difficult exercises. In order to keep up the motivation of the students, a learning unit should not exceed 15 min processing time, although students' attention levels vary widely based on factors like motivation, emotion, enjoyment and time of day (Bunce, Flens, and Neiles 2010).

The authors believe that teaching computer programming is different from teaching other subjects. Programming languages have syntax rules, which need to be memorised in order to become an efficient programmer. A capability for algorithmisation is another important factor. Software engineers need to break down a problem into a set of instructions that can be executed by the computer. The nature of programming is that in most cases there are several almost equal solutions: the same algorithm can be implemented in a number of different ways.

Without an automatic evaluation system, teachers and/or teacher assistants would have to execute the submissions and grade manually based on extensive check lists. Moreover, it is essential to give a novice programmer instant feedback on the quality of their (repeated) submission that cannot be performed without an automatic grader. Besides, it may also occur that teachers or teacher assistants do not understand correct student solutions (after looking at dozens of similar programmes) or they grade similar programs differently. In the case of complex tasks, it is impossible to do check lists that cover all possible cases. Also applications that actually work on the student's machine can fail on the teacher's machine.

To eliminate these problems, MOOC systems apply automatic graders to assess the thousands of programmes submitted by the students of the programming course. Creating a fast and accurate feedback in the context of MOOCs is a real challenge.

The purpose of the grading subsystem is to evaluate the code written by the students no matter what version they come up with. Consequently, a grader is extremely useful for software programming courses where learners are asked to submit complex code blocks. An efficient and flexible grader is generally an external subsystem of a MOOC framework. Basically, it should consist of fully functioning compilers that can execute the code submitted by the students. Some factors to consider when designing a grader for computer programming courses are:

- The grader should be able to compile the code. It is not sufficient if a grader simply compares the code as text with a predefined answer.
- The grader should run the student's code separately. The execution of the codes of Alice and Bob must be totally independent.
- The execution of malicious code or badly written code should not affect the MOOC system. So when Alice writes her first endless loop, the MOOC system should continue responding. And when Bob instructs the computer to wipe out the hard drive, that code must not be executed on the live system.
- Efficiency is an important factor. Thousands of people will submit their code at the same time. The grader must put the requests in a queue and respond as quickly as possible.
- The grader should support virtual environments. You may want to host your MOOC system on a UNIX-like system but teach Windows programming. The external grader preferably runs in a virtual environment.
- Security is paramount. Some students may be earning credits for their degree, and they should not be able to hack their own grades.

Citation: Research in Learning Technology 2017, **25:** 1945 - http://dx.doi.org/10.25304/rlt.v25.1945

- The answer from the grader should clearly indicate the acceptance of the code, or in case of errors, it should provide hints to resolve the problem.
- The grader should always maintain anonymity; it must not be revealed to any participant whether the code belongs to Alice or Bob.

The assessment of submitted programmes in MOOC systems

There are several sites that offer various online courses including computer programming, for example, Coursera, Codeacademy, Udacity or edX.

Coursera is a venture-backed, for-profit, educational technology company that offers MOOCs. They offer courses from several dozen universities and other organisations in different subjects (Coursera 2016). Coursera also offers Computer Science courses but the use of auto-graders is not typical in their programming courses. To assess students' skills they prefer to use multiple choice and true or false questions.

Compared to other online education platforms, Codecademy focuses entirely on giving coding lessons to students. Courses include web fundamentals, PHP, JavaScript, jQuery, Python, Ruby and APIs. Students can also combine these languages for their projects. The courses of the site are offered for novice programmers and codes can be written in the code panel and evaluated either line by line, or the output of the programme is tested applying a black-box test.

Udacity is a for-profit educational organisation offering MOOCs. Their main focus is on vocational courses for professionals such as Front-End Developer or iOS Developer. Each course consists of several units comprising video lectures with closed captioning in conjunction with integrated quizzes to help students understand concepts. Programming classes use the Python language; for programming assignments, the submitted codes run in a full environment using a sandboxed grading system on Amazon Web Service (AWS). All the students' codes run in this environment, which allow them to offer a much wider array of programming languages/environments from within the browser than would otherwise be possible. Courses do not offer dynamically changing exercises and the programming courses do not test programmes that use the GUI framework (Courses and Nanodegree Programs 2016).

In May 2012, recognised universities such as Harvard University and Massachusetts Institute of Technology (MIT) began to offer free online courses, assessments and certificates of completion for a fee. Their initiative was called edX and was based on an open-source software technology platform. In the open edX platform, a tool called Studio is used to create the structure of the courses and to add course content, including exercises (called 'problems'), videos and other resources for learners. Another important component of this platform is the open edX Learning Management System (LMS). This provides facilities for participants to access course content, including videos, textbooks and problems, and to check their progress in the course. To assess students' codes, an external grader can be built and deployed separately from the edX platform.

Using a third-party grader tool is a feasible option (Guillaume 2015); see, for example, the INGInious tool, which provides smooth integration with open edX and offers a safe environment for running the submitted programmes (Usage of INGInious 2016).

The external grader in edX

The MeMOOC project MeMOOC stands for Miskolci Egyetem (University of Miskolc, in English) and MOCCs (MeMooc System 2015). It is a social learning portal based in open edX that has been developed at the University of Miskolc, Hungary. MeMOOC has 12 flagship modules which provide course material on information and communication technologies (ICT), servers, computers and programming and multimedia development (MeMooc System 2015). They are as follows:

- Web application development
- Database systems and applications
- Systematic approach to IT systems
- Information and communication technologies
- Multimedia content creation
- Mobile devices programming
- Introduction to programming languages
- Computer networks and security
- Building computer networks
- Server administration

There are 54 courses altogether related to subjects for the full-time IT students at the university. Each course is available in Hungarian and in English as well. For programming courses, an advanced grader tool has been developed that analyses the code written by the students and evaluates the code. In this way, students get instant feedback on the code they have written without the constant presence of a teacher (Kusper *et al*. 2016).

The grader evaluates programme code; it is that simple. The improvement made for MeMOOC was that the code is accompanied by predefined test cases and the result seen by the students is the output from the unit tests. In this concept, course developers write unit tests for the programming exercises. If all the test cases are passed, then the submitted code is correct. If a unit test fails, then the code is not correct. It can provide hints to the students on how to fix the typical errors.

MeMOOC, based on edX, includes external graders that receive the learner responses to a problem, process those responses and return feedback. Graders run as a service and can be built and deployed separately from the edX platform. The edX platform communicates with the external grader through an interface called XQueue. It transmits learners' input to the grader; then it asynchronously waits for results from the grader and returns them to the LMS. Submissions are collected by a message queue, where they are stored until the grader actively retrieves, or pulls, the next submission from the queue for grading. The external grader polls XQueue through a RESTful interface at a regular interval. When the external grader retrieves a submission, it runs the tests on it, then pushes the response back to XQueue through the RESTful interface. XQueue then delivers the response to the edX LMS. This is how an active grader works, by asynchronously pulling messages from a queue and pushing answers back.

A passive grader, on the contrary, waits for XQueue to send the task to be checked and works synchronously. The LMS sends messages that are handled by a queue. XQueue checks the settings and determines which URL can be associated with which task. The message will be delivered to the proper URL. The passive grader

Citation: Research in Learning Technology 2017, **25:** 1945 - http://dx.doi.org/10.25304/rlt.v25.1945

gets a POST request from XQueue and a response in sync. The answer for the LMS will be delivered by XQueue through the proper URL. Figure 1 represents a simplified model of the external grader in edX.

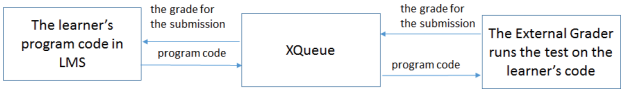

Figure 1. Model of the external grader process in edX.

The uploaded programme code is stored in a database. By default, open edX allows large files to be uploaded to an Amazon S3 bucket and saves the URL of these into the local database. This approach is very useful if there are thousands of file submissions. The downside is that Amazon S3 is a commercial service, thus adding some maintenance cost, and causing external dependency on the service. In MeMOOC, a local MySQL database was established to store all submitted files.

The external grader of MeMOOC includes a messages queue (RabbitMQ) that works as follows: (1) It receives the message from XQueue. (2) RabbitMQ reads data from the database and (3) serialises it. Then (4) MeMOOC sends an HTTP message to the external grader and (5) waits for the response. In the meantime, the external grader (6) initialises a virtual environment to (7) interpret/compile and execute the unit test that verifies the submitted code. (8) The output of the tests is saved to a file having a random filename.

Java graders in MeMOOC

What are the key factors of a successful automated grader system? According to Hundley and Britt (2009), one of the most important parts of a successful programming course is a good assignment. Ala-Mutka (2005) declared that the use of automatic tools requires more methodological attention in the settings of assignment and assessment. Hollingsworth (1960) stated that assignments need to be properly formulated in order for automatic assessment to be effective. Douce *et al*. (2005) warns that the specification of programming tasks for an automated grader always requires more precision than for the equivalent manually graded assignment, and it should not be allowed to contain ambiguities especially when considering input/output formats.

Determining whether a programme works properly is not an easy task; the theoretical limits are discussed in Beckman (1980). While there are some mathematical theories for checking the correctness of an algorithm, these are complicated and not easy to implement; moreover, the computer programming courses of MeMOOC suggest students should test their programmes using test files and unit tests. Ala-Mutka (2005) remarked that the test case design highly influences the coverage of the assessment. To avoid the false identification of programmes, Montoya-Dato *et al*. (2009) suggest that the set of test cases needs to be 'well thought out'. It is also particularly important to select test cases covering all paths throughout the programme. The test data are the Achilles' heels of any system that applies automatic graded system as Pieterse (2013) states.

Applying test cases is not sufficient for checking the code and the programming styles. For novice programmers, learning the correct coding style is a basic requisition. The applied assessments in programming courses for beginners must guarantee that students learn coding conventions: proper variable names, the insertion of comments

in the code, etc. Thus, an automatic grader must be available for analysing the submitted code, line by line.

Portals that wish to offer programming courses in which GUI classes of the programming language need to be applied must provide an automatic grader that can assess these programmes. In this case, applying a simple unit test or a grader that can analyse the code line by line is not enough.

When programmes are automatically assessed, students may tend to submit someone else's work. To prevent plagiarism for courses that offer credits is also an objective of an assessment system.

That is the reason, that in MeMOOC, different grading methods are in use for learning Java programming. The most frequently used one is based on the external grader. When grading a Java code submission, the first step is the compilation of the student's source code and the second one is the execution of the compiled code by means of unit test. In case of any compilation error, the process stops and the error message of the compiler will be sent between <pre> html tags to the LMS, which displays the message as it is. (For graders developed for other languages, this may be different but the logic is the same.)

When the grader runs the code, both the memory consumption and the execution time are limited by a virtual environment. By default, Java virtual machines (JVM) assign 32MB memory for a virtual machine; if a programme exceeds the memory limit, then an exception is thrown and execution stops. Besides the memory limit,

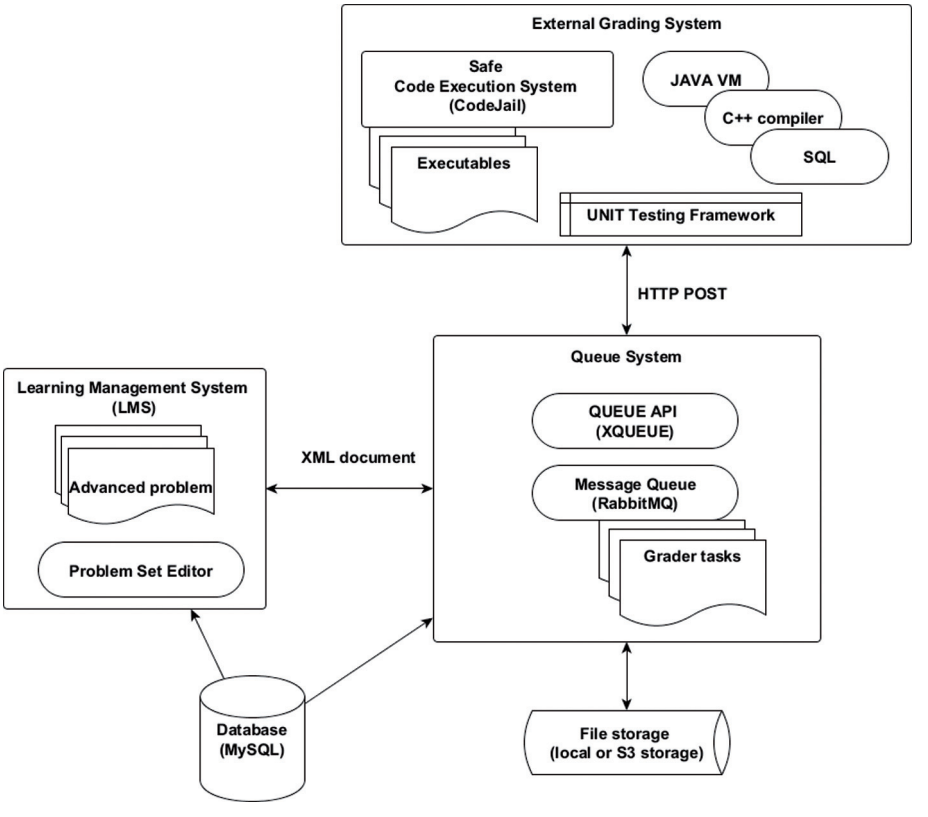

Figure 2. Evaluating coding assignments by means of an external grader.

Citation: Research in Learning Technology 2017, 25: 1945 - http://dx.doi.org/10.25304/rlt.v25.1945

it is expedient to limit CPU time. A configuration setting allows the maximum processing time to be defined so that endless loops or inefficient algorithms would result in an out-of-time exception.

XQueue is triggered by a timer to check whether there is an unprocessed item in the queue. If it finds one, then it processes the item and sends back the result to the LMS. The response should be delivered in a maximum of 4–5 s in order for students not to become impatient. If a student navigates away from the page, the answer will be registered in the background, so as not to disappear. Figure 2 presents how the external grader works in edX.

In the traditional education system, there are various other communications between the instructor and the student; therefore, feedback is paramount in MOOCs.

The implementation of the unit test–based grader

MeMOOC executes a unit test in the submitted Java code. This requires JDK7 and JUnit4 to be installed on the server. Simple methods and full programmes can be graded in programming courses. Using the assertEquals method of JUnit, the response of the student code is compared with the expected result. After running test cases and recording results for a submission, the grader returns information by posting a JSON response.

The JSON string contains a value that indicates whether the submission was correct, the score and the message to display, for example:

```
{
"correct":true,
"score":1,
"msg":"<p>The code passed all tests.</p>"
}
```

```
  <?xml version="1.0" encoding="UTF-8"?>
<problem>
<script type="loncapa/python">
  >>> the place of the dynamic python code
</script>
  >>> The text of the task in HTML
  <coderesponse queuename="advanced-grading-queue">
>>> quename attribute denotes the name of the message
>>> the answer can be determined either in an editor or by an uploaded
file)
<textbox mode="java" tabsize="2" />
  <codeparam>
  <answer_display>
  >>> in case of required help this text will be displayed
  </answer_display>
  <initial_display>
  >>> here is the code for the student
  </initial_display>
  <grader_payload>
  >>> JUNIT source code that performs the tests
  </grader_payload>
  </codeparam>
  </coderesponse>
  </problem>
```

Figure 3. The XML skeleton of a coding task.

If the programme code is not correct, it is also possible to inform the student which test case the submitted code failed on and why the programme code is not correct.

The assignment is represented by an XML file internally. This format allows the system to be extended for external graders. Figure 3 represents the skeleton of the XML file in which a problem can be coded.

The unit test must be placed in the <*grader_payload*> section of the XML file that also contains the text of the task as well. Figure 4 shows a screenshot of a task and a text box in which the student can code or copy the programme code.

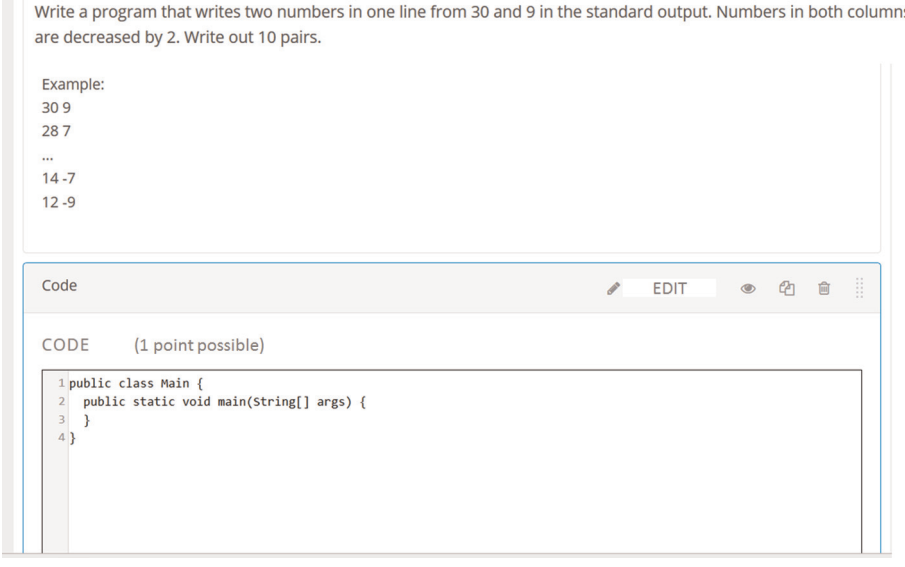

Figure 4. The coding assignment in MeMOOC.

Examining the structure of the skeleton code, it is easy to recognise that there is a section that contains the text of the problem. If the xml file is preprocessed by means of a Python script, then the coding problem becomes dynamic. In the example presented above, the constant numbers of the task are actually random numbers that are generated when initialising the coding assignments. In the XML file, Python variables are used, see <*script*> section, and these variables are embedded in the XML file. This method ensures dynamically changing tasks to eliminate plagiarism. Figure 5 shows the dynamic text of the task.

The code segment can be placed in the section of <*initial_display*> <*/initial_display*>

The unit test must be implemented in the section of <*grader_payload*> <*/grader_payload*>

Figure A.1 in the Appendix for codes represents a unit test.

Please note that the unit test contains code bits that are not compliant with Java syntax. Unit test case codes can contain special Python variables like $num1, $num2 and $num3 providing dynamic behaviour. The system evaluates the Python variables when generating the assignment. Using this technique, students will get assignments with random parameters. If authors set up a library of coding exercises, then Python code can pick and render a random coding assignment.

Citation: Research in Learning Technology 2017, **25:** 1945 - http://dx.doi.org/10.25304/rlt.v25.1945

```
<?xml version="1.0" encoding="UTF-8"?>
<problem>
<script type="loncapa/python">
<![CDATA[
  num1 = random.randint(30, 200)
  num2 = random.randint(30, 200)
  num3 = random.randint(1, 4)
</script>
<p>Write a program that prints a pair of numbers separated by a space between $num1 and $num2 to the standard output. In the following line decrease both numbers by $num3. Write out 10 pairs.</p>
    <p>Example:<br />$num1 $num2<br />$num1-$num3 $num2-$num3<br />...<br />$num1-$num3*8 $num2-$num3*8<br />$num1-$num3*9 $num2-$num3*9</p>
    ...
```

Figure 5. The dynamic text of a task.

Furthermore, Python evaluates expressions like ($num1–$num3*8). As a result, the coding assignments will have great flexibility. So MeMOOC can generate:

- the same coding assignment for all the users
- coding assignments with random assignment parameters
- random coding assignment

After the Python preprocessor runs it, the resulting code will be standard Java code that has constant values. When running the grader, the Java compiler will cope with the unit test code, thus evaluating the student's submission.

The dynamic assessment of learners' code

As can be seen in the example above, MeMOOC can generate assignments for the students. This section covers the methods that are used to assess student code. There are at least four different aspects to consider:

(1) the syntactical aspect
(2) the coding conventions aspect
(3) the structural aspect
(4) the business logic aspect.

The syntactical aspect

The syntactical aspect means that the code provided by the student will be compiled/interpreted by a compiler/interpreter. The syntactical rules of the coding language being used must be followed.

The coding conventions aspect

The coding conventions aspect covers the process to check whether students have followed the coding guidance they are given. See, for example, the following assessment (Figure 6):

Create an instance of Scanner class called kb. Declare a byte variable called sz in the second line. In the third line, get some value from the keyboard and put to that variable.

```
   ...
   <script type="loncapa/python">
   def vglcfn(e, ans):
    return ans.replace(" ","") ==
"Scannerkb=newScanner(System.in);bytesz;sz=kb.nextByte();"
   </script>
   <customresponse cfn="vglcfn">
        <jsinput
        gradefn="getGrade"
        width="900"
        height="200"
        html_file="/static/textarea.html"
        />
   </customresponse>
</problem>
```

Figure 6. The simple coding style aspect of the grader.

The xml of the assignment is given above. The grader code

return ans.replace(" ","") ==
"Scannerkb=newScanner(System.in);bytesz;sz=kb.nextByte();"

works as follows: it removes the spaces from the answer code and checks if the answer is exactly the same as expected. This approach requires a very detailed assignment description. The graders implementing this type of assessment are typically used in the first weeks of the coding courses.

As mentioned above, this method is not preferred, so an advanced method was used to check coding style. Checkstyle is a highly configurable tool to help professional programmers write Java code that adheres to a predefined (Sun or Google) coding standard (Checkstyle – code style analysis tool for Java 2016). It defines a set of modules, which can examine common styling mistakes, for example, wrong naming conventions of classes, methods and attributes; presence of mandatory headers; and improper use of imports, scope modifiers, instruction blocks, long line lengths, complexity measurements. In Figure A.2 in the Appendix for codes, curly bracket formatting problems are demonstrated as an example. Google Java Style Guide (2016) defines the proper usage of braces, and the following code violates it. Running coding style check from a shell, Checkstyle detects these problems; see Figure A.3 in the Appendix for codes.

The structural aspect

The structural aspect comes in when the students can create complex data/programme structures and these need to be checked. For example:

Declare a class called Person. Declare a string member variable called name. Implement a method called getName() that gets the name of the person. Have an integer member called age.

Citation: Research in Learning Technology 2017, **25:** 1945 - http://dx.doi.org/10.25304/rlt.v25.1945

To check a code that implements the example above, we have to check whether:

- the class called Person was declared
- the member variable called age was declared
- the method called getName() exists

The grader of this code uses the well-known JUnit code. It tries to load the Person class dynamically. If it does not exist, it throws a ClassNotFoundException exception, which will be passed back to the student. The existence of the getName() method is also checked, and if it is missing, then the grader throws an assertion failure, in other words its unit test fails. If the method requires some arguments to be passed then the person.getMethod call has to contain the list of parameters.

Figure 7 shows an example how to check member variable 'age'. The code gets the fields of 'Person' class, and the corresponding unit test fails if 'age' is not found.

```
public class TestPerson {
_____________________________________________________________________________

@Test
 public void test() throws ClassNotFoundException {
  Class person = ClassLoader.getSystemClassLoader().loadClass("Person");


  try {
   person.getMethod("getName");
  }catch (NoSuchMethodException e) {
   assertFalse("getName() not found", true);
  }

  Field[] fields = person.getDeclaredFields();


try {
   person.getField("age");
  }catch (NoSuchFieldException e) {
   assertFalse("age field not defined", true);
  }
 }
```

Figure 7. Grader: the structural aspect.

The business logic aspect

Standard JUnit tests were used to check business logic. See the following example:

```
 @Test
  public void testRetirementAge() {
   Person p = new Person();
   p.setAge(60);
   assertEquals(p.getAgesUntilRetirement(), 5);

   p.setAge(70);
   assertEquals(p.getAgesUntilRetirement(), -1);
  }
```

Figure 8. Example of simple business logic grader.

Implement a function that calculates how many years are left to work until a person can retire (normal retirement age is 65). To test the code that implements this logic, the following unit test can be used (Figure 8).

To avoid failure owing to not producing the correct output in the expected format by the submitted programme, we tried to simplify the output format (programmes display only numbers instead of long texts combined numbers).

Testing programmes with GUI in MeMOOC

Although command-line interface (CLI) applications may be sufficient for improving programming skills, graphical user interfaces (GUI) in applications would increase student motivation and experience (Staubitz *et al*. 2015). Usually, students are interested in learning how to develop programmes having GUIs (English 2004). Likewise, applications that perform 3D rendering or animations attract learners.

Programmes that use Java GUI frameworks cannot be tested efficiently by integration tests. Consequently, there is another way of testing and grading students' code. Developing a software test environment for programmes involving GUI items is ranked beyond the typical abilities of students and educators (Thornton *et al*. 2008). According to Thornton, one response to this problem is educational GUI libraries, such as those presented in English (2004) and Thornton *et al*. (2008).

In the external grader system developed for MeMOOC, testing Java programmes with GUI items uses the Mockito framework (Mockito mock testing framework 2016). For example, let's assume the coding assignment is to add three checkboxes and implement their event handlers. In Mockito, the test is implemented as checking if three checkbox objects are instantiated and the addItemListener method is called three times. To create an external GUI, grader for the Java, Mockito and Powermock-module-JUnit are installed on the server side. In the test programme, the existence of the class is checked in the following way: our grader tries to refer to an object that must be created by the student's programme and if it does not exist in a catch block, the grader catches the error. Based on the exception caught, the external grader can inform the student what is missing in the submitted programme. The response may contain tips on how to resolve the error caught.

Evaluating the effectiveness of the proposed solution

At the time of the starting requirement analysis for MeMOOC, there was no appropriate support for dealing with coding submissions. The initial question was the feasibility of extending the edX framework with this feature. Since edX is a general MOOC framework, there was less support for special features of learning to code online. The biggest achievement of this project was to design and implement the extension of the edX's basic grading system with code submissions. We have successfully established flexible templates and best practices, and created coding exercises, which are capable of checking syntactical, semantical and pragmatic aspects of submissions. The finer the feedback we want to provide, the more preparation the time required for creating coding exercises. In case of our Java programming courses, about 60% of course development time was spent on preparing coding exercises.

Citation: Research in Learning Technology 2017, **25:** 1945 - http://dx.doi.org/10.25304/rlt.v25.1945

Using this approach to coding submissions, we found it possible to generate dynamic problems in MeMOOC, not only for the coding submissions but also for all problem types.

In dynamic problems, the text of the exercises and solutions can be dynamically generated. In this way, creating graded tests require less effort from tutors, because rather than needing to create a high number of tests, they can create a single master test that appears to be a random exercise to students. At the time of writing this article, more than 17 000 Hungarian students signed for 1 of our 150 MeMOOC courses. The courses containing dynamic coding problems have 600% more subscriptions than the average.

How does MeMOOC affect traditional higher education?

The authors are working as lecturers for Hungarian Universities and we have compared and contrasted traditional and MeMOOC-based code submissions. In general, creating a new code submission (with its possible solutions), requires about 60 min. In a 20-person group, we need to spend 50 min with each student to submit a coding exercise. It takes 6 h of work in total. With MeMOOC, it takes 6 h to create the new code submission (with a number of code checks).

So, having more than 20 students, the invested effort would be worth it. Therefore, MeMOOC contains not only open courses for the public but also a high number of courses that extend the number of traditional higher education subjects.

Discussing the pros and cons of this approach and wider application potentials

The difficulty with our approach is the long preparation time necessary to create the coding exercises. But once the submissions are created, they can serve any number of students. Our automated grader provides qualitative feedback; for example, if the student's programme output does not produce the expected result or if a common error was made, it can provide a textual hint sufficient to tell the student what is wrong with the solution.

The advantage of our grading system as explained is that it can be applied for programming languages other than Java. The disadvantage is that, each new programming language requires that its own compiler be installed and configured properly.

Conclusions

In this paper, the concept of evaluating programming submissions of MOCCs was presented. The method was implemented for the Java programming language but can be extended to any programming language that supports unit testing. Four layers of code assessment were identified, demonstrating how to check the programming style, the code structure, the business logic and GUI codes. Some examples for each aspect were presented. In addition, the dynamic generation of coding assessments was implemented, with Python scripts being used for this purpose. These methods were implemented in Java programming courses that can be accessed at www.memooc.hu

Acknowledgement

This research was partially supported by the European Union and the Hungarian State, co-financed by the European Regional Development Fund in the framework of the

GINOP-2.3.4-15-2016-00004 project, aimed to promote the cooperation between the higher education and the industry.

References

Ala-Mutka, K. (2005) 'A survey of automated assessment approaches for programming assignments', *Computer Science Education*, vol. 15, no. 2, pp. 83–102.

Beckman F. S. (1980) *Mathematical foundations of programming*, Addison-Wesley, Boston, 1st edition (June 1980). ISBN-10: 020114462X.

Bunce, D. M., Flens, E. A. & Neiles, K. Y. (2010) 'How long can students pay attention in class? A study of student attention decline using clickers', *Journal of Chemical Education*, vol. 87, no. 12, pp. 1438–1443. doi: 10.1021/ed100409p

Checkstyle – code style analysis tool for Java, 2016, [online] Available at: http://checkstyle.sourceforge.net

Coursera, 2016, [online] Available at: https://www.coursera.org/

Courses and Nanodegree Programs, 2016, [online] Available at: https://www.udacity.com/courses/all

Douce, C., Livingstone, D. & Orwell, J. (2005) 'Automatic test-based assessment of programming: A review', *Journal on Educational Resources in Computing*, vol. 5, no. 3, p. 4.

English, J. (2004) 'Automated assessment of GUI programs using JEWL', *ACM SIGCSE Bulletin*, vol. 36, no. 3, pp. 137–141.

Google Java style guide, 2016, [online] Available at: https://google.github.io/styleguide/javaguide.html

Guillaume, D., *et al.*, (2015) 'Automatic grading of programming exercises in a MOOC using the INGInious platform', *Proceedings Papers*, p. 86, [online] Available at: https://www.info.ucl.ac.be/~pvr/DervalEMOOCS2015.pdf

Hollingsworth, J. (1960) 'Automatic graders for programming classes', *Communications of ACM*, vol. 3, no. 10, pp. 528–529.

Hundley, J. & Britt, W. (2009) 'Engaging students in software development course projects', *The Fifth Richard Tapia Celebration of Diversity in Computing Conference: Intellect, Initiatives, Insight, and Innovations (TAPIA '09)*, ACM, New York, pp. 87–92.

Joo, Y. J., Joung, S. & Kim, E. K. (2013) 'Structural relationships among e-learners' sense of presence, usage, flow, satisfaction, and persistence', *Educational Technology & Society*, vol. 16, no. 2, pp. 310–324.

Kusper, G., *et al.*, (2016) 'Introducing MeMOOC and recent results in e-learning at University of Miskolc', *Proceedings of 'WOW! Europe Embraces MOOCs'*, Rome, pp. 75–78.

MeMOOC system, 2015, [online] Available at: http://memooc.hu

Mockito mock testing framework, 2016, [online] Available at: http://mockito.org/

Montoya-Dato, F. J., Fernández-Alemán, J. L. & García-Mateos, G. (2009) 'An experience on Ada programming using online judging', *Reliable Software Technologies – Ada-Europe 2009, 14th Ada-Europe International Conference, Brest, France, June 8–12, 2009*, Proceedings (Ada-Europe), Springer, London, UK, pp. 75–89.

Pieterse, V. (2013) 'Automated assessment of programming assignments', *Proceedings of the 3rd Computer Science Education Research Conference on Computer Science Education Research*, ACM Digital Library, New York, pp. 45–56.

Robins, A., Rountree, J. & Rountree, N. (2003) 'Learning and teaching programming: A review and discussion', *Computer Science Education*, vol. 13, no. 2, pp. 137–172.

Staubitz, T., *et al.*, (2015) 'Towards practical programming exercises and automated assessment in Massive Open Online Courses', *2015 IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE)*, IEEE, New York, pp. 23–30.

Thornton, M., *et al.*, (2008) 'Supporting student-written tests of GUI programs', *ACM SIGCSE Bulletin*, vol. 40, no. 1, pp. 537–541.

Usage of INGInious, 2016, [online] Available at: http://inginious.org/#usage

Vihavainen, A., Luukkainen, M. & Kurhila, J. (2012) 'Multi-faceted support for MOOC in programming', *Proceedings of the 13th Annual Conference on Information Technology Education*, ACM, New York, pp. 171–176.

Citation: Research in Learning Technology 2017, **25:** 1945 - http://dx.doi.org/10.25304/rlt.v25.1945

Appendix for codes

```
public class GraderUnitTestSample {

    ByteArrayInputStream in;
    ByteArrayOutputStream out;
    PrintStream originalOut;
    InputStream originalIn;

    String myCase = "10";
    String mySol = "";

    @Before
    public void setUpStreams() {
    originalIn = System.in;
    out = new ByteArrayOutputStream();
    originalOut = System.out;

        System.setOut(new PrintStream(out));
        System.setErr(new PrintStream(out));
    }

    @Test(timeout=1000)
    public void test1() {
        Main.main(null);
        for (int i=0;i<10;i++)
           mySol+=($num1-$num3*i).toString()+($num2-$num3*i).toString();
        assertEquals(mySol,
              out.toString().replaceAll("\r", "").replaceAll("\n", ""));
    }

    @After
    public void nullStreams() {
        System.setIn(originalIn);
        System.setOut(originalOut);
        System.setErr(originalOut);
    }

    public static void main(String[] args) {
        Result result = JUnitCore.runClasses(Grader2.class);
        for (Failure failure : result.getFailures()) {
            String myFailure = failure.toString();
            myFailure = myFailure.replace("&lt;", " ");
            myFailure = myFailure.replace("&gt;", " ");
            System.out.println(myFailure);
        }
        System.out.println(result.wasSuccessful());
    }
}
```

Figure A.1. Grader unit test example.

```
public class Person
 {  // Violates 4.1.2 point of [25]: No line break before the opening brace.
   private Date birthDay;
   public Date getBirthDay()
   { // the same formatting problem
     return birthDay;
   }
   public void setBirthDay(Date birthDay) {
     this.birthDay = birthDay;
   }
 }
```

Figure A.2. Advanced coding style aspect of the grader.

```
Starting audit...
[WARN] Person.java:2:1: '{' at column 1 should be on the previous line. [LeftCurly]
[WARN] Person.java:5:3: '{' at column 3 should be on the previous line. [LeftCurly]
Audit done.Audit done.
```

Figure A.3. Checkstyle output.

Citation: Research in Learning Technology 2017, **25:** 1945 - http://dx.doi.org/10.25304/rlt.v25.1945