Automatic Grading System for Object-Oriented Programming on an E-learning Platform

Darwin Poveda, Gabriel Lopez Fonseca, Jorge Carvajal, Franklin Sánchez

Abstract— **Manual grading of Object-Oriented Programming assignments implies that sometimes the feedback arrives late. In addition, the main grading criteria is usually functionality, which leaves other grading criteria aside. This paper presents a platform that supports different automatic grading processes with quick feedback. Therefore, this work integrated an architecture that supports different grading metrics with the Edx MOOC. By using the external grader, which is a component of Edx, the student´s answer code was sent and feedback was received. The results demonstrate the feasibility of integration of Edx with the Grader.**

Keywords— automatic grading; e-learning; oop

I. Introduction

Nowadays, Object-oriented Programming is required for some technical engineering degrees. However, the students face difficulties to learn programming, and it could cause to fail the course [1]. Therefore, this teaching-learning process becomes a challenge to teachers and students.

Previous investigations have claimed that the increase in the amount of exercises to student, and a quick feedback, is essential to improve the learning process [2]. On the other hand, the increase in the number of exercises and many students in the engineering classes makes a manual grading process unfeasible. For this reason, several investigation have been oriented to develop technology to perform automatic grading with quick feedback in the process of teaching-learning [3] [4] [5].

In [6], several automatic grading programming assignments tools are reviewed. It was divided in two categories: mature, and recent developed tools. The analysis of the two types of tools shows improvements in this field of research, including security, more linguistic support and plagiarism detection. In addition, the lack of grading metrics for evaluation is a major gap in the revised tools. In [7] a typology of the automatic grading tools was established and relevant information was provided, such as considering a temporal evolution of these tools.

A massive Open online course (MOOC) is a way to connect, collaborate and learn by using a browser as a virtual class. This is an online course, which offers learning material to share with people around the world [8]. There are many platforms, such as Moodle, Canvas, Sakai, Blackboard, Coursera, Udacity, and Edx. Many universities like Harvard and MIT, and other companies like Google and Microsoft supported the development of MOOCs, which made them to gain importance in the context of higher education [9].

This project used the Grader proposed in [10], in order to support different grading metrics with the MOOC Edx. To begin, a review was performed about the MOOC platforms currently used to analyze their characteristics. Then, Edx was selected as the base platform for this project. For integration, it was necessary to analyze each component, layer and software element of the architecture of the Grader proposed in [10]. Eclipse IDE and Java programming language were used to verify the functionality of the Grader, and the required corrections were performed. Following, by using the software development methodology Extreme Programming (XP), the software artifacts needed for the creation of the Grader integration module with Edx were designed. Then, the MOOC Edx was deployed and configured on a server and the software artifacts designed were developed. Finally, this module was deployed in the Edx platform and a set of tests were carried out to verify its functionality, and corrections were made.

The development of this work has been divided into five sections. It is detailed in section II the analysis of components used for the proposed investigation. Next, in section III it is explained the integration of the Grader with the platform Edx. Finally, in section IV there are presented the conclusions of the project.

II. Analysis of Components

A. Grader Architecture

The architecture of the Grader proposed in [10] is based on the use of the Orchestration Service [11]. It has some important characteristics:

The authors gratefully acknowledge the financial support provided by the Escuela Politécnica Nacional, for the development of the project PII-15-16.

Darwin Poveda is with the Department of Telecommunications and Information Networks, Escuela Politécnica Nacional, Quito, Ecuador, (darwin.poveda@epn.edu.ec).

Gabriel Lopez Fonseca is with the Department of Telecommunications and Information Networks, Escuela Politécnica Nacional, Quito, Ecuador, (gabriel.lopez@epn.edu.ec).

Jorge Carvajal is with the Department of Telecommunications and Information Networks, Escuela Politécnica Nacional, Quito, Ecuador, (jorge.carvajal@epn.edu.ec).

Franklin Sánchez is with the Department of Telecommunications and Information Networks, Escuela Politécnica Nacional, Quito, Ecuador, (franklin.sanchez@epn.edu.ec).

- Complete Control of the Processes, service calls, and implements the grading process.
- Uses an XML configuration file to define the grading process.
- Uses calls and answers to the submodules of the grading process.

This architecture supports several grading criteria or metrics, which are considered grading submodules independent of each other. This independence helps to provide modularity, extensibility and flexibility.

Figure 1 shows the layer-based architecture. The two top layers are fixed and the three bottom layers are completely dynamic.

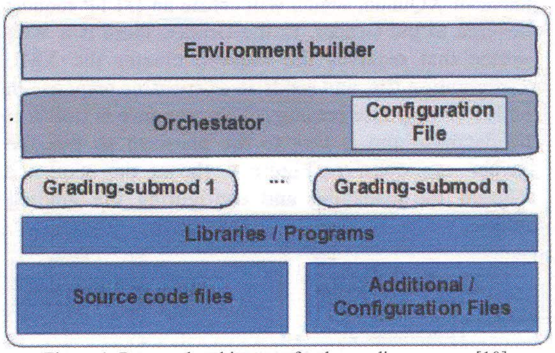


Figure 1. Proposed architecture for the grading process [10].

It has four grading submodules: the first checks the structure of a set of files (CheckGradingSubmodule). The second compiles a set of source code files in Java (CompilationGradingSubmodule). The third, tests a set of source files against test cases (TestGradingSubmodule). Finally, the forth evaluates the style of a source code file in Java (StyleGradingSubmodule).

B. *Edx external grader Module*

Edx has a service called XQueue, which allows to communicate Edx with the Grader [13]. XQueue exchanges information through JSON objects. For this job, this module must be modified to add more queues. The queues are used to support: grading submodule administration, sending additional files, and submitting assignments to be evaluated. A queue is a socket that has an IP address and a destination port. The web service, which connects with each queue is located within the Grader and it receives JSON objects.

The following steps describes the entire process of external grader when evaluating an assignment:

- The student edits the solution code for an assignment in an external grader Textbox. Then select Submit and wait for feedback.
- The Grader takes the submitted code from XQueue.
- The tests that the teacher has defined are executed in the Grader.
- The Grader returns the final grading and comments from the assignment to XQueue.
- XQueue delivers the results to the Edx Learning Management System (LMS) and is stored in the database.
- The student receives the final grade and feedback of the assignment.

III. Integration of the Grader with Edx

It is necessary to create a mechanism to integrate the architecture of the Grader with the Edx platform. The features added to the platform, after adding the grading module are:

- Administration of grading submodules.
- Administration and configuration of the grading process.
- Automatic grading, and feedback considering the grading submodules.

A. *Selection of the platform*

It was selected Edx as a base platform for the Grader because it has the following characteristics [12]:

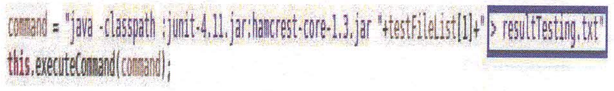
- Flexible architecture, this means that the platform architecture allows to communicate with other components. For example, an automatic Grader.
- GNU/GPL license. Allows to modify and improve design for specific needs.
- Easy access to documentation and source code.
- Provides a component called external grader.

B. *Grader Customization*

The Grader proposed in [10] was created to work on the Moodle platform with the Virtual Programming Lab (VPL) module. For this research, the Grader was given some corrections for integration with the EDX platform using the external grader module. These corrections are described below:

- In the CheckGradingSubmodule.java class, a compressed file (Zip extension) was expected as a student response, in which all files must be included to be qualified. In this project, the submissions are made by writing the response code in a Textbox of external grader, therefore, the lines of code whose functionality was to receive a compressed file and the decompression of it, were commented.
- In the TestingGradingSubmodule.java class, the Corrector.java file is used, which has test cases to evaluate the code sent by the student. To compile the Corrector.java file you need the tool JUnit [13]. This tool allows to create test cases, making instances of the classes to test their correct functioning. Its result is the number of correct and incorrect tests, which are stored in a text file. This last process of the grading submodule had errors. To solve this, after compiling the Corrector.java file using two JUnit executable JAR files, a statement was raised to save the result to the

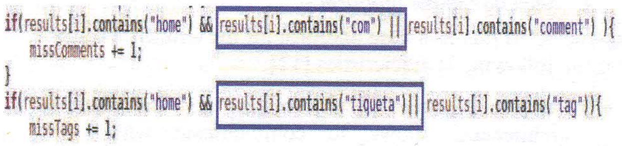
resultTesting.txt file. This file will be created in the same directory where the Grader is located, as shown in Figure 2. The number of correct and incorrect tests will be extracted from this file, with which the grade of the submodule is calculated.



```
command = "java -classpath :junit-4.11.jar:hamcrest-core-1.3.jar "+testFileList[1]+"> resultTesting.txt"
this.executeCommand(command);
```

Figure 2. Corrective in the TestingGradingSubmodule.java class

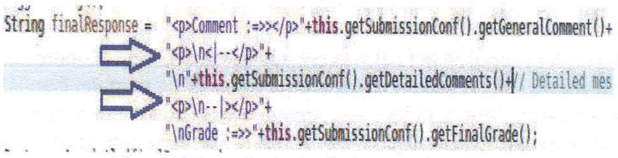
- In the StyleGradingSubmodule.java class, the Checkstyle [14] tool is used. This tool allows you to verify that a Java source code file (.java extension) has all the necessary comments and tags, in Javadoc [15] format. This tool results are the location of the comment or the label missing of the code being grading. It was found that the comment counter and missing tags in the code for the Spanish language did not work correctly. The problem was solved by adding a logical operator OR (||) within the conditional, in order to additionally look for matches in the Spanish language, as can be seen in Figure 3



```
if(results[i].contains("home") && results[i].contains("com") || results[i].contains("comment") ){
    missComments += 1;
}
if(results[i].contains("home") && results[i].contains("tiqueta")|| results[i].contains("tag")){
    missTags += 1;
```

Figure 3. Corrective in the StyleGradingSubmodule.java class

- Finally, in the Orchestator.java class, the HTML tag was increased to define a paragraph (<p> </p>), as shown in Figure 4. This change allows to generate an organized result at the time of displaying the comments to the student at the Edx platform.



```
String finalResponse =  "<p>Comment :=>></p>"+this.getSubmissionConf().getGeneralComment()+
                        "<p>\n<|--</p>"+
                        "\n"+this.getSubmissionConf().getDetailedComments()+// Detailed mes
                        "<p>\n--|></p>"+
                        "\nGrade :=>>"+this.getSubmissionConf().getFinalGrade();
```

Figure 4. Corrective in the Orchestrator.java class

After the customization, it was tested its operation using a scenario created in the Eclipse IDE. After performing all these configurations, the Grader was exported as an executable JAR file. Its name is Evaluation.jar and it was stored in a GitHub repository, it can be observed or downloaded from [16].

C. *Integration Design of the Grader with the Edx platform*

The integration can be described through a set of processes: create an assignment, manage and configure the grading process, and submit an assignment. Following, there are explained each integration process design.

- Create an assignment: First the teacher must access to Edx Studio. Then it is created an assignment using the external grader component and send it to the platform studio. The assignment is stored in the database and returns an affirmative response. The assignment is created in the Edx Studio and LMS. Figure 5 shows the sequence diagram to create an assignment.

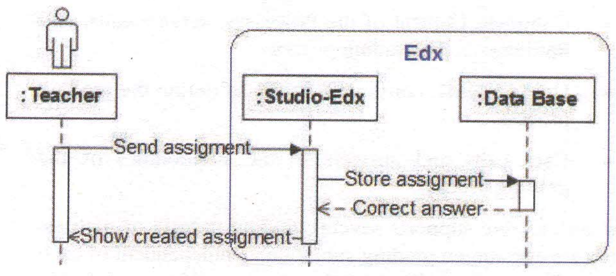


Figure 5. Sequence diagram to create an assignment

- Manage and configure the grading process: The teacher must manage the grading submodules and send to the LMS of the EdX platform. This configuration is stored in the database. It is then sent to the socket submission-queue of XQueue, which will create an HTTP request message to the Grader. In the Grader, there is a Web service that receives the request, creates the XML configuration file, and sends an affirmative response in the HTTP response message. This response is stored in the database and is sent to the platform so that the teacher can observe. Figure 6 shows the sequence diagram for managing and configuring the grading process.

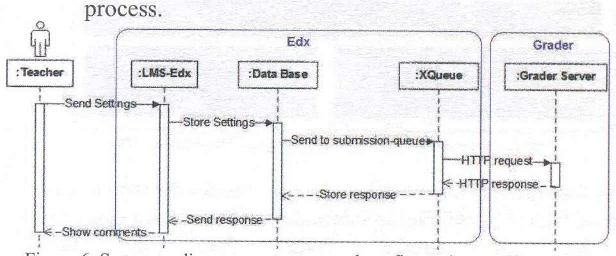


Figure 6. Sequence diagram to manage and configure the grading process

- Submit an assignment: The student must edit their Java source code for response to an assignment, using an external grader text box and send to the Edx platform LMS. The response code is stored in the database. It is then sent to the Java-queue of XQueue, which creates an HTTP request message to the Grader. In the Grader, there is a Web service that receives the request, begins the grading process, and sends the final grading and comments in an HTTP response message. This response is stored in the database and is sent to the platform so that the student can observe the assignment's feedback. Figure 7 shows the sequence diagram for submitting an assignment.

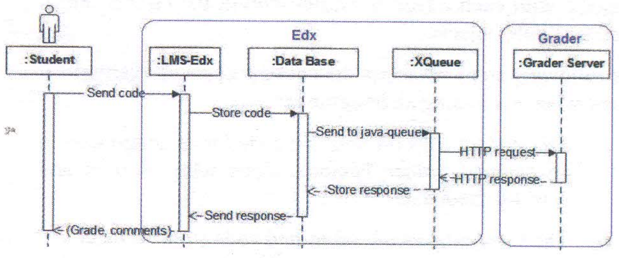


Figure 7. Sequence diagram to send an assignment

D. Implementation of the integration of Grader with Edx

After installing Edx (v Ficus. 3) [17] in the Ubuntu Operating System (v 14.04), some configurations were made, because to use the Grader it is necessary that all the required files are within it. For this project, in XQueue of the Edx platform, three new queues were created: submission-queue, which communicates with the SubmisssionConf.py service to create the XML configuration file; Corrector-queue, which communicates with the Corrector.py service to create the Corrector.java file; and Java-queue, which communicates with the JavaGrader.py service to start the grading process. To configure the three new queues, the following steps were followed:

1. Edit the xqueue.env.json file. The following three lines were added:

```
"java-queue": "http://localhost:1710",
"corrector-queue": "http://localhost:1720",
"submission-queue": "http://localhost:1730",
```

After this step, three new queues have been added to XQueue. To start the Grader the following steps were followed:

1. Access to the /edx directory and download the GitHub evaluation Project from [16].

   The project has the files Submission.py, Corrector.py, JavaGrader.py, Evaluation.jar and another files necessary for the Grader.

2. Access the /edx/Evaluation/ directory and run the three Python files in the background.

```
sudo python JavaGrader.py &
sudo python Corrector.py &
sudo python SubmissionConf.py &
```

3. Verify that all three services are running in the background.

At the end of this point, the Grader was working and waiting for an assignment to be qualified.

The operation of the JavaGrader.py and SubmissionConf.py files, which are necessary for integration, is detailed below.

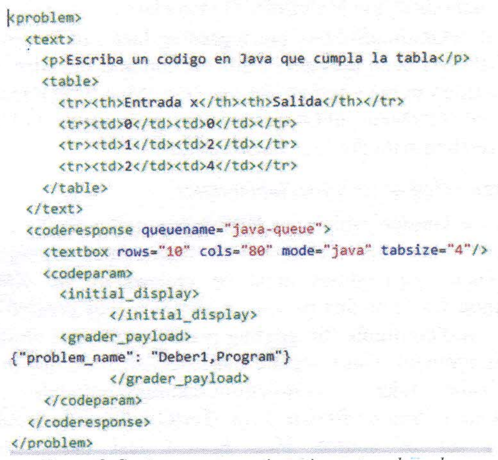
The executable file JavaGrader.py, receives from Xqueue all the information of the assignment sent in a JSON object, begins the grading process and returns the feedback of the assignments. BaseHTTPServer of Python 3, was used to implement an HTTP server (Web server). The following steps describe the operation of the JavaGrader.py file:

- An HTTP server is created, which listens to all client requests in the socket (IP address: localhost, port: 1710).
- The sent response Java source Code, the package name, and the name of the Java files to be created are extracted from the JSON object.
- With the above information, all Java source (.java extension) codes are created within the indicated package.
- The grading process is executed by sending the XML configuration file as a parameter.
- The final grading and comments returned by the Grader are captured.
- The Java source codes (.java) and bytes source codes (.class) are deleted from the package. The source codes were created by JavaGrader.py service, instead the byte codes were created by the Grader.
- Creates a JSON object that contains the state of the process, the final grade, and the comments. This object is sent as an answer to XQueue.

The SubmisssionConf.py execution file, receives from XQueue a JSON object with all the configuration of the grading process. Using that object creates an XML configuration file. Like the JavaGrader.py file, Python 3 BaseHTTPServer was used to create a Web server. The following steps describe the operation of the SubmisssionConf.py file:

- An HTTP server is created that listens to all client requests in the socket (IP address: localhost, port: 1730).
- The entire grading process configuration, the package name and the name of the XML file to be created are extracted from the JSON object.
- With the above information, the XML configuration file is created within the indicated package.
- Creates a JSON object that contains the state of the process and a successful creation comment from the XML configuration file. This object is sent as an answer to XQueue.

Following, a scenario has been created in Edx using the external grader component. Figure 8 shows the basic settings for this component.



```
<problem>
  <text>
    <p>Escriba un codigo en Java que cumpla la tabla</p>
    <table>
      <tr><th>Entrada x</th><th>Salida</th></tr>
      <tr><td>0</td><td>0</td></tr>
      <tr><td>1</td><td>2</td></tr>
      <tr><td>2</td><td>4</td></tr>
    </table>
  </text>
  <coderesponse queuename="java-queue">
    <textbox rows="10" cols="80" mode="java" tabsize="4"/>
    <codeparam>
      <initial_display>
            </initial_display>
      <grader_payload>
{"problem_name": "Deber1,Program"}
            </grader_payload>
    </codeparam>
  </coderesponse>
</problem>
```

Figure 8. Setting up an exercise using external grader.

This configuration uses HTML tags. The fields of the configuration are:

- Text: this field contains the problem statement.

- Coderesponse: this field contains the queue name, to which a JSON object is to be sent. This object contains all the information in the submission and the response code edited by the student.
- Textbox: this field defines the size of the text box and the style (either text, Java, XML, etc.) so that the student can edit his code. If there are many files, they must be separated with the tag ¨*Codigo¨.
- Codeparam: this field contains the parameters for sending the student's response. This in turn has two fields:
    - Initial_display, in this field you can edit an initial text that appears in the TextBox.
    - Grader_payload, this field contains a JSON object to create all the files to qualify. The name of the object is problem_name and the content is the name of the package and the files to be created for your grading. The package to be used and the names of the files must be separate with commas (,) and without spaces.

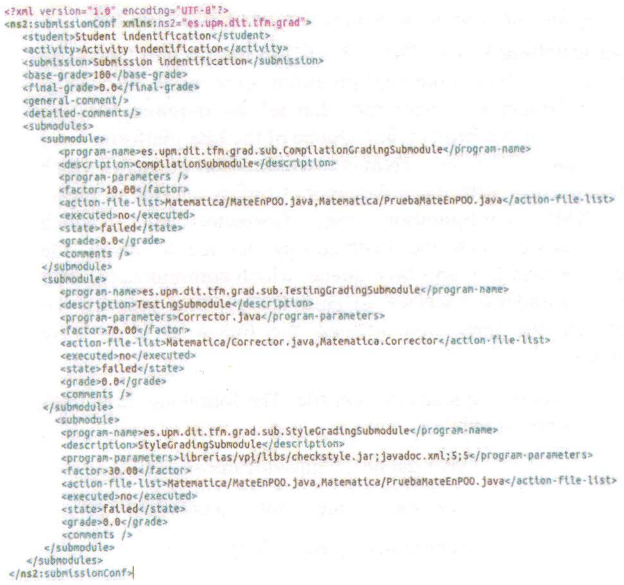
E. *XML Configuration File*

To execute the Grader's performance tests is necessary the XML configuration file that is is shown in Figure 9. For this scenario, this file has 3 grading submodules:

- CompilationGradingSubmodule, in this section in the <factor> tag a grading factor of 10 out of 100 has been defined and in the <action-file-list> tag the files to be compiled are written (MateEnPOO. java and TestMateEnPOO.java).
- TestingGradingSubmodule, in this section in the <factor> tag a grading factor of 70 out of 100 has been defined and in the <action-file-list> tag the file Corrector.java is written, which runs tests on the methods of the MateEnPOO.java class.
- StyleGradingSubmodule, a grading factor of 20 out of 100 has been defined in this section and the files are written in the <action-file-list> tag (MateEnPOO.java and TestMateEnPOO.java) that your style will be verified with the file checkstyle.jar.

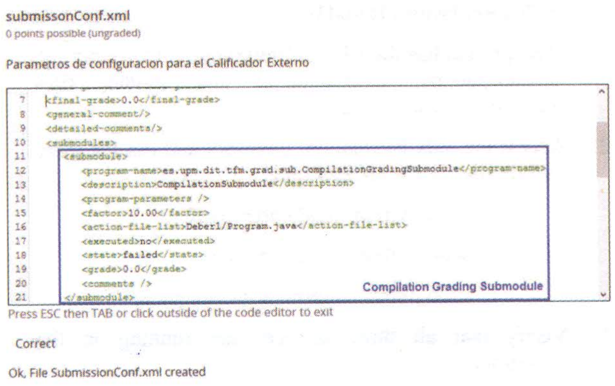
F. *Configuration of Grading Submodules*

To use the Grader, which has been integrated with the Edx platform, it is necessary to create all the submodules of grading to use. These submodules must be created in the XML configuration file. For this project, a template was created to administer and configure the grading process, which is unique for each assignment. This template was established for the three grading submodules: CompilationGradingSubmodule, to compile a set of source files in Java; TestGradingSubmodule, to test a set of source files against test cases; and StyleGradingSubmodule, to evaluate the style of a source code file in Java. This template is only visible by the teacher or administrator of the platform. The template format for Compilation Grading Submodule is shown in Figure 10, which applies to the other three submodules since they have a common format.



```xml
<?xml version="1.0" encoding="UTF-8"?>
<ns2:submissionConf xmlns:ns2="es.upm.dit.tfm.grad">
   <student>Student indentification</student>
   <activity>Activity indentification</activity>
   <submission>Submission indentification</submission>
   <base-grade>100</base-grade>
   <final-grade>0.0</final-grade>
   <general-comment/>
   <detailed-comments/>
   <submodules>
      <submodule>
         <program-name>es.upm.dit.tfm.grad.sub.CompilationGradingSubmodule</program-name>
         <description>CompilationSubmodule</description>
         <program-parameters />
         <factor>10.00</factor>
         <action-file-list>Matematica/MateEnPOO.java,Matematica/PruebaMateEnPOO.java</action-file-list>
         <executed>no</executed>
         <state>failed</state>
         <grade>0.0</grade>
         <comments />
      </submodule>
      <submodule>
         <program-name>es.upm.dit.tfm.grad.sub.TestingGradingSubmodule</program-name>
         <description>TestingSubmodule</description>
         <program-parameters>Corrector.java</program-parameters>
         <factor>70.00</factor>
         <action-file-list>Matematica/Corrector.java,Matematica.Corrector</action-file-list>
         <executed>no</executed>
         <state>failed</state>
         <grade>0.0</grade>
         <comments />
      </submodule>
        <submodule>
         <program-name>es.upm.dit.tfm.grad.sub.StyleGradingSubmodule</program-name>
         <description>StyleGradingSubmodule</description>
         <program-parameters>librerias/vpl/libs/checkstyle.jar;javadoc.xml;5;5</program-parameters>
         <factor>30.00</factor>
         <action-file-list>Matematica/MateEnPOO.java,Matematica/PruebaMateEnPOO.java</action-file-list>
         <executed>no</executed>
         <state>failed</state>
         <grade>0.0</grade>
         <comments />
      </submodule>
    </submodules>
</ns2:submissionConf>
```

Figure 9. XML configuration file



submissonConf.xml
0 points possible (ungraded)

Parametros de configuracion para el Calificador Externo

```xml
<final-grade>0.0</final-grade>
<general-comment/>
<detailed-comments/>
<submodules>
    <submodule>
        <program-name>es.upm.dit.tfm.grad.sub.CompilationGradingSubmodule</program-name>
        <description>CompilationSubmodule</description>
        <program-parameters />
        <factor>10.00</factor>
        <action-file-list>Deber1/Program.java</action-file-list>
        <executed>no</executed>
        <state>failed</state>
        <grade>0.0</grade>
        <comments />
    </submodule>
```

Press ESC then TAB or click outside of the code editor to exit

Correct

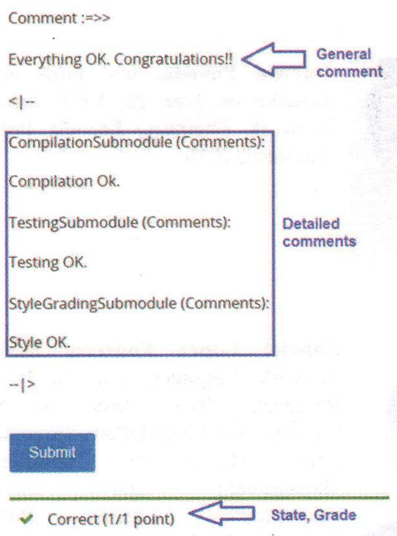
Ok, File SubmissionConf.xml created

Figure 10. Compilation Grading Submodule

After creating an activity with assignments at Edx, it must be published before it can be visualized in the platform. It can be configured the grading process, by increasing or decrementing submodules from the template. If it is set the grading process, then it must be set the factor field considering each submodule used.

G. *Feedback from the Grader*

The process for the student is very simple. First, it has to be edited the solution code for an assignment. Then it is selected Submit, and finally it is received the feedback from the Grader. An example of feedback is shown in Figure 11.


Comment :=>>

Everything OK. Congratulations!!

<|--

CompilationSubmodule (Comments):

Compilation Ok.

TestingSubmodule (Comments):

Testing OK.

StyleGradingSubmodule (Comments):

Style OK.

--|>

Correct (1/1 point)

Figure 11. Feedback from the Grader

IV. Results

A. *Functionality Tests and Acceptance of Students*

First, a short introduction was made regarding the use of the system to the students of the Object Oriented Programming (OOP) subject, who collaborated with the project.
Students were asked to use the Automatic Grader tool for a month, in which the solution was tested and any inconvenience was corrected. Finally, the solution was stable and we proceeded to conduct a survey to students in order to understand its user experience to determine the degree of functionality, and the acceptance level among the students. It is necessary to mention that students were asked to consider only the Inheritance and Exceptions exercises.
Important aspects were considered to determine the functionality of the solution, such as: the operation of each of the tools that make up the solution and access to each of them. Regarding the acceptance of the solution, the presentation of the solution interface, page load times, ease of use and student experiences when using the solution were considered. The results of the applied survey to students are presented below:

- The presentation of the Automatic Qualifier interface was regular.
- Regarding the time it takes to load a page within the Automatic Qualifier at the Laboratory, such as: login, online evaluation, and navigation between pages, it was obtained that the largest number of responses is in a waiting time of 2 -4 seconds. According to [10] these values are within the acceptable time range for page loading.
- Most students have little difficulty using the tool and its use was simple.
- The results obtained show that uploading the code showed difficulty, because students have a resistance to using a template to complete the solution code for a given problem.
- The results obtained show that the feedback given by the tool was useful and practical.
- The results obtained show that there is no great acceptance by the students of the style rating metric.
- According to the previous analysis, it is evident that the use of the solution was a simple and presented no difficulties for the students. Therefore, it can be concluded that the Automatic Grader had a positive impact and achieved good acceptance in the students.

B. *Functionality and Acceptance Tests of the Teacher*

The applied survey collected the opinions given by the collaborating teacher of the project. Once this information was collected, an analysis was carried out, the results of which are presented below:

- The design and ease of use, as well as the access time to the Automatic Grader is good. These results affirm the result obtained by the students.
- The display of the exercise description and the evaluation of the Automatic Grader always worked.
- The metrics that the teacher consider important are the test and compilation cases, instead the style metric leaves it aside.
- Regarding the use of the Automatic Grader in a full semester, the teacher strongly agree that it would be useful for learning of OOP.
- Finally, the teacher is satisfied with the metrics proposed and the feedback obtained from the Automatic Grader.

V. Conclusion

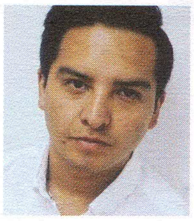
It was configured an automatic grader for Object Oriented Programming code using the Edx platform. A scenario was created to test the performance of the Grader and the required corrections were made. These corrections were carried out on the classes CheckGradingSubmodule.java, StyleGradingSubmodule.java, TestingGradingSubmodule.java, and Orchestator.java. Then it was designed the integration between the Grader and the Edx platform for the processes: creation of an assignment, management and configuration of the grading scheme, and submission of an assignment. Following this, the integration was implemented, based on the processes designed. Finally, the results of the implementation showed that the Automatic Grader checked Object Oriented Programming code by giving a score with its corresponding feedback, and the teacher was able to configure the metrics of the grading process successfully.

References

[1] S. Willman, R. Lindén, E. Kaila, T. Rajala, M.-J. Laakso, y T. Salakoski, «On study habits on an introductory course on programming», *Comput. Sci. Educ.*, vol. 25, n.º 3, pp. 276-291, jul. 2015.

[2] M. Amelung, K. Krieger, y D. Rösner, «E-Assessment as a Service», *IEEE Trans. Learn. Technol.*, vol. 4, n.º 2, pp. 162-174, abr. 2011.

[3] H. Le, «Interactive Computer Science Exercises In edX», 2016.
[4] T. Barrios y M. B. Marín, «Aprendizaje mixto a través de laboratorios virtuales», *Signos Univ.*, 2014.
[5] J. C. Rodríguez-del-Pino, E. Rubio Royo, y Z. Hernández Figueroa, «A Virtual Programming Lab for Moodle with automatic assessment and anti-plagiarism features», 2012.
[6] J. C. Caiza y J. M. del Álamo Ramiro, «Programming assignments automatic grading: review of tools and implementations», en *7th International Technology, Education and Development Conference (INTED2013)*, Valencia, Spain, 2013, pp. 5691-5700.
[7] M. Guerrero, D. S. Guamán, y J. C. Caiza, «Revisión de Herramientas de Apoyo en el Proceso de Enseñanza-Aprendizaje de Programación», *Rev. Politécnica*, vol. 35, n.º 1, p. 84, feb. 2015.
[8] P. Ruiz Martín, «Presente y futuro de los Massive Open Online Courses (MOOC): Análisis de la oferta completa de cursos de las plataformas Coursera, EdX, Miríada X y Udacity.», 2013.
[9] J. S. Ruiz, H. J. P. Díaz, J. A. Ruipérez-Valiente, P. J. Muñoz-Merino, y C. D. Kloos, «Towards the Development of a Learning Analytics Extension in Open edX», en *Proceedings of the Second International Conference on Technological Ecosystems for Enhancing Multiculturality*, New York, NY, USA, 2014, pp. 299–306.
[10] J. C. Caiza, «Automatic Grading of Programming Assignments: Proposal and Validation of an Architecture», ESPAÑA/Escuela Técnica Superior de Ingenieros de Telecomunicaciones-Universidad Politécnica de Madrid/2013, 2013.
[11] P. Mayer, A. Schroeder, y N. Koch, «MDD4SOA: Model-driven service orchestration», en *Enterprise Distributed Object Computing Conference, 2008. EDOC'08. 12th International IEEE*, 2008, pp. 203–212.
[12] «About Open edX | Open edX Portal». [En línea]. Disponible en: https://open.edx.org/about-open-edx. [Accedido: 01-jun-2017].
[13] «JUnit - About». [En línea]. Disponible en: http://junit.org/junit4/. [Accedido: 11-jul-2017].
[14] «checkstyle – Javadoc Comments». [En línea]. Disponible en: http://checkstyle.sourceforge.net/config_javadoc.html. [Accedido: 11-jul-2017].
[15] «How to Write Doc Comments for the Javadoc Tool». [En línea]. Disponible en: http://www.oracle.com/technetwork/articles/java/index-137868.html. [Accedido: 11-jul-2017].
[16] «DarwinPoveda/Evaluation», *GitHub*. [En línea]. Disponible en: https://github.com/DarwinPoveda/Evaluation. [Accedido: 14-jul-2017].
[17] «3.4.1. Installing Open edX Fullstack — Installing, Configuring, and Running the Open edX Platform documentation». [En línea]. Disponible en: http://edx.readthedocs.io/projects/edx-installing-configuring-and-running/en/latest/installation/fullstack/install_fullstack.html#installing-open-edx-fullstack. [Accedido: 03-jul-2017].



Darwin Poveda. was born in Quito, Ecuador on June 28, 1991. Information Network Engineer, Escuela Politécnica Nacional, 2018.

Gabriel Lopez Fonseca. Information Network Engineer, Escuela Politécnica Nacional, 2010. Master in Systems Security, Sheffield Hallam University, UK, 2015. Master in Communications Management and Information Technologies, Escuela Politécnica Nacional, 2017. Currently working as a Teacher Researcher at Escuela Politécnica Nacional in Quito.

Jorge Carvajal, was born in Quito, Ecuador on August 2, 1984. Electronics and Telecommunications Engineer, Escuela Politécnica Nacional. Master's degree in Information Technology, Universidad de Ciencias Aplicadas Mannheim. Currently working as an Assistant Teacher at Escuela Politécnica Nacional in Quito.

Franklin Leonel Sánchez Catota, Electronics and Telecommunications Engineer, Escuela Politécnica Nacional. Interuniversity Master in Telematic Engineering, Universidad Carlos III of Madrid and Universidad Politécnica de Catalunya. Currently working as a full-time added Teacher at Escuela Politécnica Nacional in Quito.