ACode: Web-based System for Automatic Evaluation of Java Code

Marcos André Pinto[1], António Luis Lopes[2]

Instituto das Telecomunicações - Instituto Universitário de Lisboa, Lisboa, Portugal
[1] a28453@iscte.pt; [2] alsl@iscte.pt

Abstract. Most current approaches for automatic evaluation of source code use input/output testing to validate student-submitted solutions. However, very few use software engineering metrics to analyze source code. With the limitations of related work in mind, we present, in this paper, our 4-stage approach for automatic evaluation of source code: i) the code is first compiled and checked for any errors; ii) the compiled code is then tested against a set of JUnit tests provided by the teacher; iii) a set of software engineering metrics is used to compare the student's solution against the teacher's solution; iv) and finally, based on the previous stages, feedback is provided to the students so they can self-evaluate and identify the areas in which they need further study.

Keywords: Student feedback, automatic evaluation, web application, java code

1 Introduction

The success of first year students in learning programming languages and concepts is usually associated with hard work and frequent testing of their programming skills so as to assess their current knowledge. During evaluation periods, teachers provide programming exercises for which the students have to present solutions. Afterwards, the teacher grades these tests individually and it is through this grading feedback that students can self-evaluate and determine the concepts that need further study. Several systems exist that provide an automatic process to evaluate students' solutions, with Mooshak [1], a web-based system designed to support programming contests, being one of the most used. Other systems like PC2 (desktop application [2]), UVa Judge and EduJudge [3] also provide similar functionalities.

These approaches are capable of evaluating source code by using simple methods that merely test the inputs and outputs of the compiled programs but fail to use more comprehensive methods that would enable the evaluation module to produce a more accurate grade for the student's solution. Our goal is to overcome these limitations and extend the functionality of typical evaluation mechanisms and thus provide an accurate evaluation of the entire program. We do so by introducing the use of software engineering metrics while comparing the student's solution with the teacher's solution. This data, combined with the input/output testing data, provides a complete view of the quality of the student's solution.

2 ACode System

The ACode system's goal is to provide a way for students to autonomously practice their programming skills and solve problems and assignments given by their teachers and have an automatic and immediate feedback on their performance. Through the web-based system, students submit their solution for the given problem, which is then analyzed in a four-stage process (see Fig. 1):

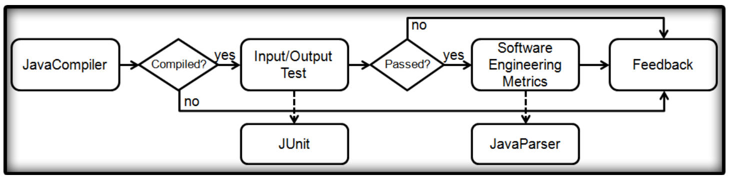

Fig. 1. ACode system's flowchart

1. The source code sent by the student is first compiled by the system to make sure that no compilation errors exist. If errors exist, then the system returns a final grade of 0 (zero). The student can then re-submit a solution.
2. If the code compiles without any errors, the second stage will run a set of JUnit tests [4] (allows a programmer to execute automatic tests on java code) provided by the teaching staff. If all tests fail, then the system returns a final grade of 0 (zero), otherwise, continues.
3. After the testing stage, the system will retrieve important data from the compiled source code (through the JavaParser framework [5]) and from the teacher's solution and compare both sets of data using simple software engineering metrics (see section 2.1).
4. Finally, the system gathers all information from previous stages and using a set of formulas (see section 2.1), calculates the final grade of the student's solution. This final grade, along with a detailed report of the evaluation is presented to the student so that he or she can assess his or her performance.

2.1 Evaluation Process

The second stage, as described in the previous section, performs a set of tests through the ACode system so as to obtain the number of successful tests in the student's solution. Expression (1) depicts the way the system assigns a grade to the test-stage of the evaluation process.

$$tests = \frac{correct\_tests \times 20}{total\_tests} \tag{1}$$

Afterwards, the system extracts specific metric-based data from both the student's solution and the teacher's solution. The number of lines of each method, the number of methods, the cyclomatic complexity [6] (measurement and control of the number of paths in a program) and the number of global variables in the class are some of the software engineering metrics that are used in the ACode's automatic evaluation module for Java source code. We have chosen this set of metrics because they are adequate to evaluate simple algorithms, which are often the ones used by first year students. The system will compare both sets of metric-based data (the student's and the teacher's), as depicted in expression (2), to calculate the grade for the metrics evaluation stage (stored in the individual_metric variable).

$$individual\_\mathrm{metric} = 20 - \mathrm{e}^{\left(\frac{\mathrm{metric}_{\mathrm{student}} - \mathrm{metric}_{\mathrm{professor}}}{\mathrm{metric}_{\mathrm{professor}}} - 1\right)} \quad (2)$$

The teacher will provide a percentage for each of the evaluation elements, which marks the relevance of that specific element within the evaluation. Table 1 describes all of the percentages used in this metrics-based evaluation.

Table 1. Description of the percentage variables

Variable	Description
P_{class}	Percentage assigned to the metrics of the class
P_{method}	Percentage assigned to the metrics of a method
$P_{project}$	Percentage assigned to the metrics of the project
pcc_{class}	Percentage assigned to the cyclomatic complexity of the class
pln_{class}	Percentage assigned to the number of lines in the class
pmn_{class}	Percentage assigned to the number of methods in the class
pan_{class}	Percentage assigned to the number of attributes in the class
pcc_{method}	Percentage assigned to the cyclomatic complexity of a method
pln_{method}	Percentage assigned to the number of lines in a method
$pcn_{project}$	Percentage assigned to the number of classes in the project
$pmn_{project}$	Percentage assigned to the number of methods in the project
$pcc_{project}$	Percentage assigned to the cyclomatic complexity of the project
$pln_{project}$	Percentage assigned to the number of lines in the project
$pan_{project}$	Percentage assigned to the number of attributes in the project

metrics_class = pcc_{class} × cyclomatic_complexity + pln_{class} × lines_number + pmn_{class} × methods_number + pan_{class} × attributes_number (3)

metrcis_method = pcc_{method} × cyclomatic_complexity + pln_{method} × lines_number (4)

metrcis_project = $pcn_{project}$ × classes_number + $pmn_{project}$ × methods_number + $pcc_{project}$ × cyclomatic_complexity + $pln_{project}$ × lines_number + $pan_{project}$ × attributes_number (5)

Expression (6) presents the formula used to calculate the grade associated with the metrics evaluation stage (stored in variable metrics) that uses the percentages described in Table 1.

$$\text{metrics} = P_{\text{class}} \times \text{metrics\_class} + P_{\text{method}} \times \text{metrics\_method} + P_{\text{project}} \times \text{metrics\_project} \quad (6)$$

Finally, all this evaluation data is combined in order to determine the final grade assigned to the student's solution. Expression (7) represents the formula used to calculate the final grade:

$$R = \begin{cases} 0 & , compilation = 0 \\ 0 & , compilation = 1 \text{ } and \text{ } tests = 0 \\ P_{r1} \times tests + P_{r2} \times metrics & , compilation = 1 \text{ } and \text{ } tests > 0 \end{cases} \quad (7)$$

In this expression, P_{r1} represents the percentage associated with the importance of the testing stage and P_{r2} represents the percentage associated with the importance of the metrics evaluation stage. The teacher provides both percentages as he (or she) sees fit. In the end, the student will get a complete report stating his or her final grade and the evaluation information that shows the tests that were successful and the ones that were unsuccessful.

Acknowledgements

This work was partly supported by Fundation of Science and Technology (FCT) under the grants PTDC/EEACRO/104658/2008, SFRH/BD/76438/2011 and PEst-OE/EEI/LA0008/2011.

References

1. Leal, José Paulo e Silva, Fernando: Mooshak: a Web-based multi-site programming contest system. In Software: Practice and Experience, Wiley Interscience, vol. 33, pp. 567-581, ISSN: 0038-0644 (2003)
2. Ashoo, Samir E, Boudreau, Troy e Lane, Douglas A.: PC2 Documentation - Version 9.1 What's New Document. Link: http://www.ecs.csus.edu/pc2/doc/v9/WhatsNewInV9.1.pdf (Accessed in 26 January 2012)
3. Revilla, Miguel A., Manzoor, Shahriar e Liu, Rujia: Competitive Learning in Informatics: The UVa Online Judge Experience. In Olympiads in Informatics, Institute of Mathematics and Informatics, Lithuania, vol. 2, pp. 131-148, ISSN: 1822-7732 (2008)
4. Beck, Kent; Gamma, Erich: JUnit. Link: http://www.junit.org/ (Accessed in 22 February 2012)
5. Project Google: Java 1.5 Parser and AST. Link: http://code.google.com/p/javaparser/ (Accessed in 11 January 2012)
6. McCabe, Thomas J.: A Complexity Measure. In IEEE Transactions on Software Engineering, Institute of Electrical and Electronics Engineers, New York, vol. SE-2, NO. 4, pp. 308-320 (1976)