2022 International Conference on Advanced Learning Technologies (ICALT) | 978-1-6654-9519-6/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICALT55010.2022.00118

LUD: An Automatic Scoring and Feedback System for Programming Assignments

Marcelo Guerra Hahn, Silvia Margarita Baldiris Navarro, Luis de-la-Fuente-Valentín
Universidad Internacional de La Rioja
La Rioja, Spain
e-mail: marcelo.guerrahahn261@comunidadunir.net, silvia.baldiris@unir.net, luis.delafuente@unir.net

***Abstract*— The increase in usage of online learning systems, has caused a renewed interest in using computers to provide more support to the student's learning experiences, allow instructors to focus on activities that require human intervention, and enable courses with a large number of students to help them achieve the learning objectives. One experience that has been particularly complex to emulate in online environments is assignment grading. The grading cycle tends to be a weak point in the experience. Among its drawbacks, two main issues are potential turnaround time and that assignments tend to provide only one opportunity to show understanding of the course content. In the context of programming assignments, this experience is particularly problematic as the programming cycle tends to be an iterative process. This paper discusses the initial implementation of an automatic scoring and feedback system for programming assignments. The system includes feedback on syntax, semantics, and code structure. We explain the architecture of the system and the results of an experiment run with 20 students that shows the effects of the system. In the experiment, we observed that the feedback system improves student performance in the assignment as measured by the grade assigned to them.**

Keywords-assessments, programming, automatic scoring, automatic feedback.

I. INTRODUCTION

Automatic scoring and feedback are the processes by which students' work is graded, and feedback is provided without the direct participation of the instructor [1]. Tools that provide this functionality have become more and more prevalent with the appearance of MOOCs and other large-scale learning environments that would not be if they required instructors to grade student submissions manually [2]. Even though the tools have become more popular lately, the basic ideas have been available for some time, starting with the usage of the Scantron to implement large-scale multiple-choice testing [2]. With the improvements in technology, the type of questions that can be automatically graded has evolved to include single-word answers [2] and short and long essays [3, 4]. Due to the structured nature of source code, computer programming is a field where automatic scoring and feedback are particularly appealing. Tools in this field tend to focus on using unit testing to verify assignments' correctness and detect plagiarism [5, 6, 7].

The positive effects of using automatic scoring have proven very useful to both institutions and students. Institutions can increase the number of students with the same number of instructors, and instructors have more time to dedicate to other tasks besides grading [8]. Students get access to faster turnaround on their submitted work and more consistency in grading [9].

When reviewing the literature in the area of automatic scoring and feedback [10], we found that most of the work focuses on scaling the ability to test students and comparing automated systems to manual mechanisms; for example Chauhan [11] shows how automatic scoring and feedback is needed to support MOOCs., [13] analyses the accuracy of a system to grade testing exercises, and [14] looks at the accuracy of a system that grades Java coding problems in MOOCs. In contrast, this work focuses on using automatic grading and feedback as a learning tool that improves student performance. We build on those ideas, focusing on driving the student towards completing the work successfully and obtaining the expected competencies. To achieve this goal, we focus on this research question:

Can students independently improve their performance programming assignments using the feedback provided by our system?

With this question in mind, we designed the *Let's Use Data* (LUD) system and ran a low-scale experiment with 20 students to tune the system in preparation for a larger experiment. This document explores the components of the system and the answers to the research questions.

A. High-level architecture

According to the literature, when working with computer source code, there are three main error points [12]:

- Compilation/Interpretation issues
- Incorrect results
- Poorly written code

As these three error points can be measured separately, it seems reasonable to design a system with separated modules to support the evaluation of each of these error points as shown in Figure 1. The system enables the instructor to set up the information needed to provide feedback and the interactions where students can receive the feedback and make use of it.

978-1-6654-9519-6/22/$31.00 ©2022 IEEE
DOI 10.1109/ICALT55010.2022.00118

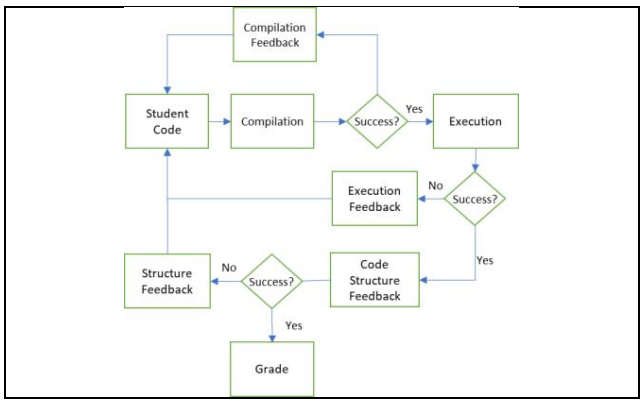

Fig. 1. Code evaluation flow diagram

B. Compilation/Interpretation

When writing code, the first step the student needs to accomplish is to make sure the code is well-written so that the computer can read it and that the code uses the correct words and places them in the proper places. Given that the computer needs to read any program the student sends, the computer produces a very generic message when the code has an issue. Messages include "error in line x," "missing x in line y," and "incorrect keyword x." In the context of a particular problem, those errors can be expanded upon so that the reader can fix them more easily. At this stage, LUD allows the instructor to expand those errors with hints based on the more likely errors and the errors students have faced. To achieve this, the system takes the student code, executes the compilation/interpretation process, collects the errors produced, then compares those errors to errors already known. If the error is already known, a hint is displayed to the student. If the error is unknown, it is displayed without a hint to the student, and the instructor is notified of the new error. The instructor then can introduce a new hint for the error.

C. Execution

Once the computer can understand the code without errors, the next step is to verify that the code produces the expected result. To do this, the system takes the student code and runs tests to verify that the output matches the expected result. If the test detects an error, the student can see it. A system similar to the one described in the previous part is used for the instructor to add hints based on the expected and actual outputs, making the problem easier to understand.

D. Code Quality

The last step in coding evaluation is to see if the code follows a set of quality parameters. Those parameters depend on the level of the course and include structure and performance. Code structure refers to the legibility of the source and it is related to basic items such as correct indentation and more subjective items such as the appropriate use of control structures and functions. On the other hand, performance is related to the resources used by the program both from a memory (how much RAM does the program uses) and execution time (how long does it take to run). For performance, the system measures the execution time of the code and returns feedback based on it. For code structure, the code compares the student's solution to a set of solutions comprised of instructor-provided solutions and instructor-selected solutions among the solutions provided by students and provides hints based on structural differences using a similar process to the one in the previous stages.

II. Evaluation

A. Methodology

To look at the system's capabilities, we evaluated its functionality in one module in a Python course. The course is designed as part of an intermediate-level Python class generally attended by students pursuing degrees in Computer Science or Data Analytics. The module teaches the usage of combinations of the Python control structures *for* and *if*. Each participant will face three problems. The first problem is a diagnostic assessment designed to create a baseline of the participant's Python programming knowledge. The second one is a new problem that does not include the feedback mechanisms, and the third one is another new problem that uses the feedback mechanisms and allows the student to resubmit as explained in section II. We will call these problems: diagnostic problem, problem with feedback, and problem without feedback.

To gather the data required for the comparison, the instructor sets up the problem and hints. The students then complete a version of the problem with and without hints. Once the students complete the work, the instructor grades all the submissions.

B. Participants

We randomly selected 20 participants using an online platform to request them. The use of the platform allowed for a selection of participants interested in the topic and represented a variety of backgrounds. Before working on the problem, participants were asked to provide information on their current level of education, understanding of programming, and current knowledge of Python. We gathered that 40% of students had no degree, 45% had a bachelor's degree, and 15% had a master's degree. We also gathered that 15% indicated no programming knowledge, 25% some programming knowledge, and 60% a high level of programming knowledge. Finally, we identified that 25% indicated no knowledge of Python, 35% some level of Python, and 40% had high knowledge of Python.

C. Metrics

To compare the student performance, the instructor manually grades the code on a scale from 0 to 10, with 0 not working and 10 being an advanced implementation in Python. For this module, the goal is to get the student to write the code at a level 7. Note that participants with previous

Python knowledge could theoretically write code at a higher level.

The students are randomly divided into two groups where one group is shown the problems in the sequence problem with no feedback followed by a problem with feedback and the other group is presented with the problem with feedback before the problem without feedback.

For answering the research question, two metrics are in use: first, the grade obtained in the exercises allows for the analysis of the students' performance. Second, the number of re-tries offers information of the students' engagement in the course and if the feedback was use.

III. Results

The diagnostic problem results shows that 25% of the participants were unable to solve it, 65% provided a solution showing a medium level of understanding, and 15% provided a solution showing a high level of understanding.

The paired t-test reveals a significative difference between the score of the participants between:

- the initial problem and problem without feedback
- the initial problem and the problem with feedback
- the problem without feedback and the problem with feedback.

The paired t-test for diagnostic versus no feedback showed that the student performance had no significant difference (p-value = 0.813). The difference between the diagnostic and problem with feedback was significant (p-value = 0.046) with a 95% confidence interval of 0.03 to 3.17, indicating a score improvement within 0.03 and 3.17. The difference between no feedback and feedback was significant (p-value = 0.008), with a 95% confidence interval of 0.41 to 2.39, indicating that the score improvement is likely within 0.41 and 2.39. Separately, we measured the number of re-tries per student and obtained an average of 2.88 re-tries implying that students could use the feedback early and submit correctly without too many interactions with the feedback system.

Looking at these results, we can see that the problem presented without feedback did not improve the student's performance compared to the baseline. In contrast, the problem presented with feedback did improve the performance. These results indicate that providing automatic feedback can help improve student performance in the given tasks. Together with this, the low number of re-tries suggests that the feedback includes sufficient information to allow the students to enhance their performance in the problem.

IV. Conclusion and Future Work

We developed a system that focuses on helping students' performance in assessments through automatic feedback. The system focuses on the three main failure points the assignment can hit, the ability to provide immediate feedback and instructor augmented feedback. The results of our experiment show that the system produces the expected results and improves students' performance.

As we are in the development stages of the system, we decided to perform a small-scale experiment to help us identify whether we are moving in the right direction. With the conclusions obtained in the current experiment, further validation requires a new experiment with more participants, more problems with and without feedback, better usability, to validate the current findings and give us more information on the effects of using the LUD system.

A. Acknowledgments

We thank PLeNTaS project, "Proyectos I+D+i 2019", PID2019-111430RB-I00 and the PL-NETO project, Proyecto PROPIO UNIR, projectId B0036 for their support.

V. References

[1] X. Dong and J. Hu, "An exploration of impact factors influencing students' reading literacy in Singapore with machine learning approaches," *International Journal of English Linguistics,* vol. 9, no. 5, pp. 52--65, 2019.

[2] Y. Cao, L. Porter, S. Nam Liao and R. Ord, "Paper or Online?: A Comparison of Exam Grading Techniques," in *2019 ACM Conference on innovation and Technology in Computer Science Education*, 2019.

[3] F. S. Pribadi, T. B. Adji, A. E. Permanasari and A. Mulwinda, "Automatic Short Answer Scoring Using Word Overlapping Methods," in *AIP Conference Proceedings*, 2017.

[4] K. Taghipour and H. T. Ng, "A Neural Approach to Automated Essay Scoring," in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Proceedings*, 2016.

[5] S. H. Edwards and M. A. Pérez-Quiñones, "Web-CAT: Automatically Grading Programming Assignments," in *13th Annual Conference on i=Innovation and Technology in Computer Science Education*, 2008.

[6] J. DeNero, S. Sridhara, M. Perez-Quinones, A. Nayak and B. Leong, "Beyond autograding: Advances in student feedback platforms," in *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, 2017, pp. 651--652.

[7] A. Wakatani and T. Maeda, "Web applications for learning CUDA programming," in *2017 8th International Conference on Information, Intelligence, Systems & Applications (IISA)*, 2017, pp. 1--5.

[8] T. Saito and Y. Watanobe, "Learning path recommendation system for programming education based on neural networks," *International Journal of Distance Education Technologies (IJDET),* vol. 18, no. 1, pp. 36--64, 2020.

[9] N. Alruwais, G. Wills and M. Wald, "Advantages and Challenges of Using e-assessment," *International Journal of information and Education Technology,* vol. 8, no. 1, pp. 34-37, 2018.

[10] M. Guerra Hahn, S. M. Baldiris-Navarro, L. de-la-Fuente- Valentín and D. Burgos, "A Systematic Review of the Effects of Automatic Scoring and Automatic Feedback in Educational Settings," *IEEE Access,* 2021.

[11] A. Chauhan, "Massive Open Online Courses (MOOCS): Emerging Trends in assessment and Accreditation," in *7-17*, 2014.

[12] Z. Ullah, A. Lajis, M. Jamjoom, A. Altalhi, A. Al-Ghamdi and F. Saleem, "The effect of automatic assessment on novice programming: Strengths and limitations of existing systems," *Computer Applications in Engineering Education,* vol. 26, no. 6, pp. 2328-2341, 2018.