

A study on how to classify exercises in introductory programming courses: a Moodle plugin contribution

Thiago Gomes Versissimo
University of São Paulo
Institute of Mathematics and Statistics
São Paulo, Brazil
thiago.verissimo@usp.br

Leônidas de Oliveira Brandão
University of São Paulo
Institute of Mathematics and Statistics
São Paulo, Brazil
leo@ime.usp.br

Ewout ter Haar
University of São Paulo
Institute of Physics
São Paulo, Brazil
ewout@usp.br

Abstract—This full paper reports the results of the development of a Moodle plugin, Programming Exercise Teaching Assistant (PETA), created to support teachers from introductory programming courses to identify problematic exercises by analyzing how students interact with them. Moodle is an open source learning management system that allows personalization of its functionality through plugins. In introductory programming courses, it is common to propose problems to students, for which they need to elaborate solution code. The VPL (Virtual Programming Lab) is a popular plugin that allows teachers to create this kind of problem, with automatic evaluation built on top of test-cases. The iAssign (interactive Assignment) is another plugin of the same sort, but with block-based programming. Our plugin is an open-source project, initially developed to integrate with iAssign, and created to integrate easily with any similar plugin used in programming tasks. Our primary goal is to empower teachers and students by supplying a tool that improves the quality of programming in education. Over the last few years, the University of São Paulo has increased its public policy for social inclusion, bringing in even more learners with diverse educational backgrounds. Understanding the students' profiles and being able to modify problems based on these new contexts is fundamental to improve the quality of teaching. This plugin aims to meet the needs of teachers and learners, exploring data from their educational context to improve the learning experience. The plugin automatically extracts data from students' interaction with the coding exercise, e.g. the time between submission (TBS), code changing between submissions (CCBS), and grades. Using these variables, we calculated derived metrics in an attempt to classify exercises according to different parameters. First, we considered time-related metrics, the highest time between submissions (HT). Secondly, we evaluated metrics related to code changes, the highest code changes between submissions (HC). Finally, we accounted for student performance by considering average grades, penalized by time and by number of submissions (AG). A technical contribution of this work is the Moodle plugin that analyzes the students' codes of iAssign, available as free software. The results from this new plugin with eight introductory classes show that it was possible to classify exercises. The AG metric created could be associated with the exercise level of difficulty, while HT and HC metrics are indicative of exercise complexity, requiring a higher level of prior knowledge, such as familiarity with loops, functions, and potential challenges with the problem instructions.

Index Terms—Learning Analytics, Formative Assessment, Moodle, Programming exercises

I. INTRODUCTION

The deliberate and focussed practice that programming exercises afford make them one of the essential components in the design of rich learning experiences for students in computer science courses. Well-designed exercises can help teach a large variety of programming concepts to students of diverse backgrounds with respect to ability or experience [1]. The quality of an exercise can be given various meanings, and in this paper we take the perspective that exercise quality must first of all be evaluated by teachers in relation to the particular student that uses them.

We start therefore from a conception of education in which the learner learns something, for a purpose, from a teacher [2]. In this conception, learning is not just growth or development, it is motivated and happens by interactions with other humans, to distinguish it from mere behavior modification. For the purposes of this paper the main interactions we are interested in are those between teachers and learners, as mediated by programming exercises and these three actors are our main units of analysis.

Ever since Seymour Papert stated that “to integrate computational thinking into everyday life was insufficiently developed” [3], computing became more and more part of our daily life and *computational thinking* is now considered fundamental to well educated citizen [4]. For this reason, introductory programming courses are becoming ever more popular [4]¹. and studies on how to teach programming have become an active field of academic endeavor [5], [6].

In this paper, we describe the first results of an effort to develop a framework that assists teachers in the design of their programming exercises for introductory programming courses. Building on previous work, we take fine-grained log data generated by the interactions of students with automatic evaluated exercises hosted in a learning management system and define metrics for exercises with respect to difficulty, complexity and other educationally relevant concepts [7]. Our framework design aims to then present these exercise quality indicators to teachers in the most effective way possible.

¹Garber reported that, at Harvard University, the “introductory computer science” (CS50) increased tenfold between 2008 and 2017.

Our first research question is: how can we operationalize the concept of “quality” of a programming exercise? Since one exercise can be useful for one but not for another student, we will have to take into account this relational aspect of exercise quality in some way. We also anticipate that the programming exercises will be mostly deployed in formative assessment contexts, which will have to guide our choice of metrics and indicators [8].

Our second research question is: how can we best assist teachers to design and deploy programming exercises in their courses? A related question is: what is the best way to present metrics and indicators about exercises and students to the teachers so they can take effective action?

We report on our action research carried out within the context of introductory programming courses offered to a diverse group of students, including both summer school students with no programming experience as well as preservice math teachers who are enrolled in introductory programming courses. In total, our research covered eight courses, comprising 182 exercises, and involving a total of 287 students. These courses are hosted on the widely-used and open source Moodle Learning Management System, with use of our own open source plugin, iAssign [9], that allow the elaboration of programming exercises with automatic code evaluation, similar to the well-known VPL plugin [10]. iAssign uses another interactive Learning Module [11], the iVProg², that is a free software³ implementing textual and visual programming [12]–[14].

Before we describe the development and validation of our programming exercise assistant, we briefly discuss traditional psychometric measures of quality of exercises for educational assessment and compare other measures to this baseline theoretical framework. We then discuss related work and our own proposal for a platform to investigate how to assist teachers in creating and validating programming exercises. The results of the development work as validated on real course data are presented, with finally a discussion and directions for future work.

II. BACKGROUND ON EDUCATIONAL ASSESSMENT AND RELATED WORK

The field of educational assessment is very mature and even if we restrict our attention to programming education or standardized test analysis, the literature is vast (see e.g. [15]–[18]). Less work is available, however, for the quantitative analysis of formative assessments. Well-known operationalizations of exercise quality (we’ll use “items” in this section) in the frameworks of Classical Test Theory (CTT) or Item Response Theory (IRT) are usually applied to summative assessments with dichotomous scoring in which the test taker gets the answer either right or wrong ([18], [19]). Although we are mostly interested in the formative use of programming exercises, it is nevertheless worthwhile to briefly review both test

analysis frameworks in order to build upon their established concepts which we can then try to extend. As examples of those traditional psychometric concepts, both CTT and IRT frameworks operationalize the concept of ability for students and difficulty and discrimination for items, but do so in different ways. Also important are the notions of reliability and validity, which are properties of tests that account for uncertainty of various types [20].

A. Classical Test Theory

The basic unit of analysis for CTT is the test, a collection of items or tasks, presented to the student. The usual way to score the test and thereby estimate the ability of the student is to simply sum the amount of correctly answered items, the sum score. This is an intuitive way to proceed and easy to interpret for non-psychometricians like teachers and students. In a CTT analysis, a more difficult item for a certain population of test-takers is one with the smaller amount of correct answers. The discrimination of an item is a measure of how well it contributes to distinguishing test-takers with less ability from those with high ability. One measure for discrimination is the so-called item-test correlation. Items that correlate only weakly with the total sum score have low discrimination, since they do not contribute to picking out weak test-takers very well.

B. Item Response Theory

One characteristic of CTT is that its characterizations of items like difficulty or discrimination are in relation to the particular test-taker population. For example, testing an (on average) less able population would result in a higher item difficulty than the same item in a test applied to a higher ability population ([21]–[23]). In IRT modeling the unit of analysis is the item, for which parameters like difficulty and discrimination are estimated on a population independent scale. Moreover, the item difficulty and the ability estimate are on the same scale, allowing for predictions like “a test-taker with ability X will have $Y\%$ chance of correctly answering this item”. Although the mechanics of this statistical modeling (the “scaling”) and the scoring (the ability estimate) are more complicated than in CTT, the interpretation of the item and student characteristics is more straightforward, since its goal is to allow for inferences about items that are independent of the test-taker population. Likewise, student ability estimates should be independent of the items used. IRT allows items to be tested in one population, put in item banks and then applied to other populations. Once the items for a certain construct have been calibrated, a population can be tested longitudinally with different tests measuring the same construct. In computerized adaptive testing items with appropriate difficulties for the test-taker are selected from an item bank, allowing for more efficient tests. All of these advantages of IRT rely on its assumptions being satisfied, in particular, the one-dimensionality of the construct “ability”. One consequence of the linear scale constructed by IRT modeling is the value put on items with high “discrimination” parameters, since they efficiently distinguish between test-takers. At the same

²<http://www.usp.br/line/ivprog>

³<http://200.144.254.107/git/LInE/iVProg>

time, this may not favor the use of these items in formative assessment contexts.

C. Quality of programming exercises

The Moodle LMS incorporates reports [24] that use these well-established psychometric frameworks, giving difficulty and discrimination parameters for dichotomously scored quiz questions. But these quality indicators for questions cannot be simply applied in the case where the items are interactive programming exercises which are deployed in formative assessments. When students keep interacting with the exercise until all unit tests pass, the notion of simple dichotomous scoring does not really apply. Even more importantly, a lot of information about the exercise and its relation to the student is contained in the exact way the solution is approached. Therefore, although the relevance of traditional concepts like difficulty, discrimination and reliability is undisputed, we should look for ways that extend them for the context of programming exercises. Finally, it is not clear that the attempt of traditional IRT to abstract exercise characteristics like difficulty or discrimination away from particular student groups is always appropriate. It may well be that indicators exist that are tailored to particular groups and that are more useful measures of exercise quality to teachers in their particular context.

D. Presentation of results and interface

The indicators and characteristics of the programming exercises that we construct may be related to traditional indicators like difficulty or discrimination, but are not calculated in exactly the same way. Therefore, care should be taken how they are presented to teachers (and maybe students) in order to minimize the risk of misinterpretation due to familiarity with these traditional measures. How indicators are named, whether they are presented in dichotomized form (e.g. "valid" or "invalid") or color coded in a certain scheme all may influence the interpretation of the indicators. Furthermore, the time and place they are presented is important if the results are to be used in effective practice. An off-line analysis with downloaded data in an external program would impede the rapid iterations an in-context interface next to the actual exercises would allow. This is one of the reasons we implemented our platform as a plugin of the popular learning management system Moodle. The tight integration between Moodle components that its plugin API provides allows us to collect the data generated by the interactions with programming exercises, analyze this data and present it to the teacher all in the same environment, allowing for rapid iterations and tight feedback loops.

III. METHODOLOGY

To answer our research questions we built a Moodle plugin of type block called Programming Exercise Teaching Assistant (PETA)¹. The purpose of PETA is to assist teachers in managing and analyzing data related to student performance in coding exercises. The plugin is specifically designed to

improve the teaching process and enhance student learning by offering teachers a comprehensive and in-depth analysis of a set of proposed coding exercises.

In figure 1 we present the first block of *iAssign/iVProg* exercises, that was used in the second week of one of undergraduate discipline used in this work. They had, respectively, 7, 5 and 4 exercises.

Week 02: 22/08-28/08 - Introduction to iVProg (inputs/outputs; type real)

To produce a practical algorithm it is essential the use of variables.

As an introduction to this concept, please, study the text

"Introduction to variables and arithmetics expressions".

Initial exercises: variables, input and output of data

[To home] Initial exercises: more variables, input and output of data

Exercises with real types: real variables and attributions

Fig. 1. An example of *iAssign* block of initial exercises in the 2022 undergraduate discipline.

The table I presents the 7 exercise titles inside the first block of *iAssign* exercises, on week 2.

TABLE I
THE FIRST SECTION OF *iVProg* EXERCISES IN WEEB 2 TO UNDERGRADUATE STUDENTS IN 2022.

Index	Exercise title
1.1	Introduction - Hello world!
1.2	Introduction - Read an integer and print its value
1.3	Introduction - Read 2 integers and print its sum
1.4	Introduction - Read 2 integers and print its product
1.5	Introduction - Read 2 integers, a and b, and print its integer division
1.6	Introduction - Read 2 integers, a and b, and print the remain of its integer division
1.7	Introduction - Calculate the mean (integer) between 2 integer numbers

The *iAssign/iVProg* platform incorporates a block-based programming approach, where students solve exercises by dragging and dropping blocks. Figure 2 showcases an example implementation of an exercise called *Total of Days* which involves determining the number of days in a given number of months, assuming each month has 30 days. The solution involves using blocks to define a variable for the number of months, reading input from the keyboard, and calculating the total days by multiplying the number of months by 30.

The Moodle platform has API that record data from student's interactions with course activities, including various types of information such as user activity, assessment results, and engagement with learning materials. The *iAssign* plugin, for instance, records the date and time of code submission, the submitted code itself, and the corresponding grade in a database for each code checking event. From these records,

¹See https://github.com/thiagogomesverissimo/moodle-block_peta

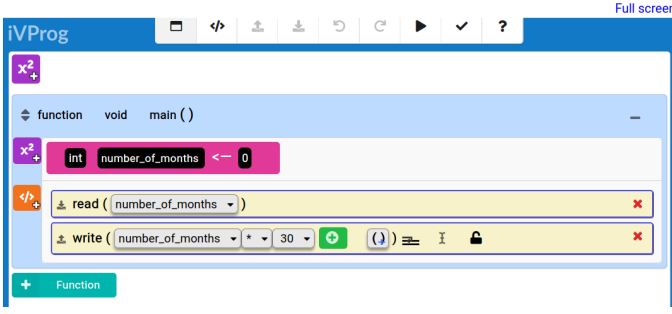


Fig. 2. Example solution to an exercise using iAssign/iVProg

variables such as the *time between submissions* (TBS), the number of *code changes between submissions* (CCBS), and the corresponding *grade* can be obtained directly from each user's submission attempts.

Based on the TBS, CCBS and grade variables recorded in the plugin table, three new metrics were derived. The first metric is referred to as the *highest time between submissions* (HT), which is calculated by comparing the time interval between sequential pairs of submissions made by a particular user for a given exercise. The pair of submissions with the longest time interval is selected as the HT for that user. We apply the same procedure for each user within an exercise. Consequently, our final table includes a row for each user, along with the corresponding HT value. Finally, the average of all HT values is computed to derive a number characterizing the exercise.

Similar to the calculation of HT, the second metric, referred to as the *highest code changing between submissions* (HC), is also derived from sequential pairs of submissions made by a particular user for a given exercise. An average of all HC values is also calculated. The code changes between submissions use the Levenshtein distance algorithm [25].

The final metric, referred to as *Average Grade* (AG), is calculated by averaging the grades earned by a particular user for a given exercise, following equation 1. This average is then penalized based on the time interval between the first and last submissions and the number of submissions made by the user. This metric takes into account the user's performance over time and their ability to maintain their code while making changes. To create these three metrics, we used previous work [7] as a reference, with some changes. For example, to ensure consistency and comparability across the metrics, we normalized all variables to vary within the same order of magnitude. We also changed some of their notation.

$$AG = \frac{\overline{\text{grade}}}{TMS + n} \quad (1)$$

$$TMS = \frac{\text{last timestamp} - \text{first timestamp}}{2} \quad (2)$$

Where n represents the number of submissions. TMS and n were normalized to remain between 0 and 1 following equation

3.

$$z_i = \frac{x_i - \min(x)}{\max(x) - \min(x)} \quad (3)$$

To avoid lengthy real-time metric calculations, our plugin creates its own database tables to store results from the calculations of the metrics. A command-line interface-based script, within the Moodle environment, is used to populate the table and can be scheduled to run automatically. By performing incremental calculations in the background, the plugin minimizes the time required for these operations and reduces the time that teachers need to wait to see the metrics. Additionally, background calculations enable the plugin to handle larger dataset environments more efficiently.

One important design goal of the PETA plugin is to present the teacher with an expedient user interface allowing for timely information about how their programming exercises are functioning for their students. The plugin includes a Moodle block that presents a list of available courses, for teachers to choose from. The interface allows teachers to select exercises from a specific course or view exercises from all courses in their Moodle instance, as depicted in Figure 3. Once a course is selected, the plugin displays relevant metrics for each exercise, facilitating interpretation of the data and enabling teachers to monitor the quality of exercises effectively. The plugin identifies the top three exercises with the highest value for each metric, as well as the three exercises with the lowest value. Teachers can then click on the exercise title to access detailed information about the exercise requirements, as we will explain in the following section.

Fig. 3. PETA Moodle block view displaying a list of available courses.

While Moodle offers numerous intelligent tutoring system plugins aimed at assisting students in their learning process [26], our plugin is more closely aligned with the concept of an intelligent teaching assistant, as defined by [27]. Specifically designed to aid teachers in managing and analyzing data related to student performance in coding exercises, our plugin focuses on improving the teaching process by helping teachers classify and improve the quality of the computing problems they propose to students. [28]

Our work is closely related to two references, [29] and [28], with [28] being written in Portuguese. In work [28], exercise classification is performed based on students' responses, resulting in the creation of a large question bank consisting

of exercises in the same category. Subsequently, exercises of similar difficulty levels are randomly assigned to students. On the other hand, work [29] also employs exercise classification based on responses but critically discusses certain limitations associated with this approach. Notably, it highlights that difficulty measures can be influenced by biases in data collection, emphasizing the importance of careful attention when measuring exercise difficulty.

In order to mitigate biases discussed in the previous work [29], our plugin implements a selection criterion that includes exercises with a minimum of 10 submissions. By doing so, we ensure that the difficulty level of exercises is accurately estimated using a sufficient amount of submission data, thereby reducing the impact of biases. These biases could potentially reflect the characteristics of a specific group.

As mentioned previously, the main conceptual reference for our work was the study referred to as [7]. However, in [7] all the analyses were conducted asynchronously, requiring the extraction of data from Moodle and performing the analyses outside of the platform. This approach demanded a certain level of data analysis knowledge and consumed the teacher's time. In contrast, our approach integrates the analysis directly into the Moodle platform, bringing it closer to the teacher's work tools. This integration simplifies the analysis process for the teacher, as they can access and view the metrics within the Moodle environment without the need for external data extraction or analysis. By offering a user-friendly interface, teachers can prioritize their teaching tasks over data management.

IV. RESULTS

In order to evaluate the concept of our plugin, we tested it within a Moodle instance comprising eight "Introduction to Programming" courses (table II). These courses specifically employed the *iAssign* tool for conducting coding exercises. Among the eight courses, six were summer courses that catered to a general audience, while the remaining two were mandatory disciplines within an undergraduate mathematics program. The summer course occurs during January and February (school holiday in the southern hemisphere), taking 6 weeks, with 5 classes per week, each one with 2 hours.

Table II provides an overview of the courses included in our study, along with their respective types, the number of exercises offered, the number of students who successfully completed the course, and the total number of submissions recorded across all exercises within each course.

TABLE II
DATA FROM EIGHT INTRODUCTION TO PROGRAMMING COURSES THAT UTILIZED THE IASSIGN PLUGIN BETWEEN 2021 AND 2023

year	type	exercises	users	submissions
2021	summer daytime	21	33	445
2021	summer nighttime	20	51	656
2021	undergraduate discipline	26	40	409
2022	summer daytime	21	32	379
2022	summer nighttime	12	38	302
2022	undergraduate discipline	36	27	414
2023	summer daytime	21	25	320
2023	summer nighttime	25	41	522

We analyzed and classified exercises in the eight courses using the derived metrics HT (highest time between submissions), HC (highest code changing between submissions), and AG (average grade). This analysis will contribute to a deeper understanding of the relationship between the calculated metrics (HT, HC, and AG) and the difficulty level of the exercises. By examining the data, we can validate the effectiveness of our procedure for evaluating and classifying the exercises.

A. Average grades, penalized by time and by number of submissions (AG)

Figure 4 showcases a key feature of the PETA plugin: a comprehensive box displaying the calculated Average Grade (AG) metric. The left side of the box presents the top three exercises with the highest AG, while the right side shows the three exercises with the smallest AG values.

Average grades, penalized by time and by number of submissions (AG)

(0.92) Total of days
(0.43) Print the greatest number among two integers
(0.37) Weighted average
(0.14) Invert a 3-digit integer
(0.14) Average consumption
(0.12) Predecessor and successor

Fig. 4. Part of the PETA interface, presenting the exercises with the highest and lowest AG metric. The normalized value of AG is presented in parentheses before the exercise title

Teachers have the option to select different courses and compare the AG values across them. Furthermore, by clicking on a specific exercise, the teacher can access a full description of the task. By presenting these metrics in a visually intuitive format, the plugin enables teachers to quickly identify exercises that require further attention or adjustments.

The exercise with the highest value of AG, 0.92, titled "Total of Days," has the following description:

In this exercise, you should implement an algorithm that calculates the total number of days in n months. For this, your program should receive an integer value, representing the total number of months. Then, it should calculate and print the result. Consider that all months have 30 days.

This exercise stands out in terms of its average grade, indicating that students performed well and achieved a high level of proficiency in solving the task.

In the right-hand columns, the exercise titled "Predecessor and Successor" has been identified as having the lowest value of the Average Grade (AG) metric, 0.12. This exercise stands out as having a lower average grade, indicating that students may have encountered difficulties or challenges in completing the task successfully.

Figure 5 shows another way to present the teacher with the indicadores for their programming exercises by way of a bar chart graph that visualizes the exercises ordered from the highest AG (Average Grade) to the smallest. Each column in the chart represents an exercise, and by hovering the mouse pointer over a column, the corresponding AG metric, exercise title, and the course it belongs to are displayed.

This bar chart should provide a clear overview of the exercises in terms of their AG values, allowing teachers to quickly identify exercises with higher or lower difficulty levels.

B. Highest time between submissions (HT) and Highest code changing between submissions (HC)

The HC and HT metrics classify exercises along a dimension which is similar to "difficulty", but to which a slightly different interpretation can be given. Again, the PETA block displays the exercises with highest and smallest values of these metrics, as well as a scatter plot and bar chart (figures 6 and 7).

Exercises with lower HC and HT values tend to be less complex and require minimal coding knowledge or implementation, as exemplified by the exercise "Determine whether 2 numbers are equal or different" with an HC value of 0.07 and HT value of 0.05. This exercise involves a simple if statement. Similarly, the exercise "Read an integer and print it" with an HT value of 0.05, requires a straightforward implementation. These exercises are considered relatively easy and suitable for beginners or introductory programming concepts.

The exercise *Array - Sum of x and y positions*, on the other hand, emerged as the exercise with the highest values in both HT (0.24) and HC (0.39) metrics, its instruction is:

In this task, build a program that receives 12 values and stores them in an array. Then, also read two numbers X and Y, corresponding to two positions in the array. Finally, your program should write the sum of the values found at the respective positions X and Y".

Indeed, exercises with high HC and HT values indicate a higher level of complexity and require a deeper understanding of programming concepts and structures. In the given example, students are expected to possess an understanding of how to manipulate array structures. So, exercises with high HT and HC values often involve the utilization of advanced programming constructs such as loops, conditionals, array manipulation, or more intricate algorithms. As we discuss below, the HC and HT indicators may represent a different dimension of difficulty than the average grade AG indicator.

To assist the teacher in their interpretation of the metrics, the plugin also generates a scatter plot that displays the relationship between the natural logarithm of the time interval

between attempts in seconds and the code difference between attempts. Each bullet represents one submission, with blue bullets indicating successful attempts and red bullets indicating failed attempts. Figure 8 presents a scatter plot generated by the PETA plugin for the exercise with the highest AG (Average Grade), while figure 9 is the exercise with the lowest AG.

The results show a positive correlation between the time interval between submission and the code difference for both exercises, as expected, since more time is needed to submit an attempt with a larger code difference. As we discuss below, we think this gives credence to our interpretation of HC and HT as indicators of complexity.

There are more blue points than red points for the exercise with the highest AG, indicating that it may be less difficult than the exercise with the smallest AG, which has a lot of red points indicating failed attempts. These findings suggest that exercises with smaller AG may be more difficult and require more attempts to complete successfully. This indicates that the AG metric has the potential to be associated with the overall difficulty of exercises in the context of the studied courses.

V. DISCUSSION AND CONCLUSION

The Programming Exercise Teaching Assistant (PETA) plugin is able to assist teachers in managing and analyzing data related to student performance in coding exercises within the Moodle Learning Management System (LMS). The developed PETA plugin addressed our research questions: *how can we operationalize the concept of "quality" of a programming exercise?* and *how can we best assist teachers to design and deploy programming exercises in their courses*, given it provides valuable features such as metrics calculation, visualization of quality indicators for programming exercises, and comparison across courses. The design of the plugin separates the definition of indicators from their presentation and to validate our design we tested the plugin with a particular set of indicator such as Highest Time between submissions (HT), Highest Code changing between submissions (HC), and Average Grade (AG).

The user interface of the plugin should allow teachers to gain insights into exercise difficulty, potential issues with instructions, and overall student performance. Using the PETA plugin with HT, HC and AG indicators on a real course data revealed interesting findings. The AG metric, which considers average grades penalized by time and number of submissions, served as an indicator of exercise difficulty. The HC and HT metrics provided additional insights into exercise complexity and potential issues with the instructions for the exercises. Exercises with higher HC and HT values were found to require advanced programming constructs and prior knowledge, while exercises with lower values indicated a more straightforward implementation. We note that the AG metric on the one hand and the HC and HT metrics on the other hand can be interpreted slightly differently, as difficulty and complexity respectively. That these meanings are not exactly the same is plausible by inspection of the exercise statements and the fact that highest and lowest ranking exercises are different for

Average grades, penalized by time and by code changes (AG)

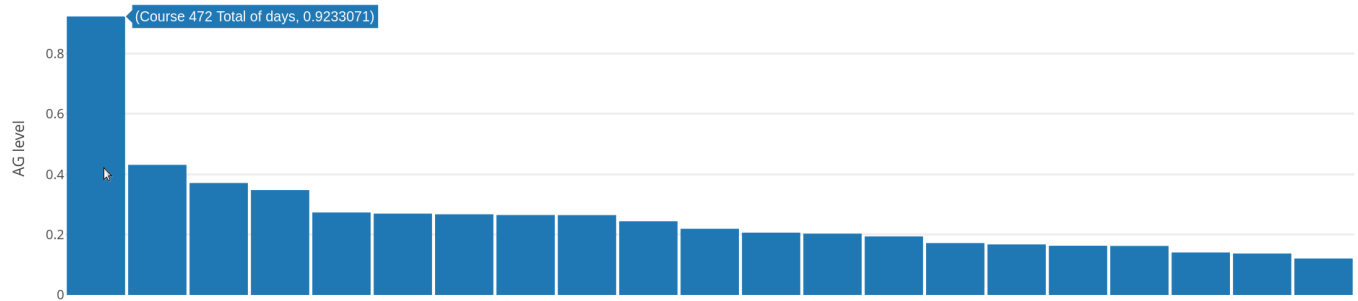


Fig. 5. Bar chart graph that visualizes the exercises ordered from the highest AG (Average Grade) to the smallest

Highest code changing between submissions (HC)

(0.39) Array - Sum of x and y positions
(0.37) Loop - Sum of odd numbers (until 0 is entered)
(0.24) Value swap
(0.06) Print the greatest number among two integers
(0.07) Determine whether 2 numbers are equal or different
(0.09) Payroll

Fig. 6. At left the 3 first exercises with highest mean of code change (HC), at right the 3 ones with smaller mean.

Highest time between submissions (HT)

(0.24) Array - Sum of x and y positions
(0.23) Average consumption
(0.22) Value swap
(0.05) Determine whether 2 numbers are equal or different
(0.05) Introduction - Read an integer and print it
(0.05) Even or odd

Fig. 7. At left the 3 first exercises with highest mean of time between submissions (HT), at right the 3 ones with smaller mean.

different indicators. It should be noted that the multidimensional character of the indicators we investigated stands in contrast to the one-dimensional scales that IRT constructs for students and items. However, from the perspective of formative assessment richer information about programming exercises could be advantageous, allowing teachers to fine-tune which exercises they offer to their students.

It is also worth noting that the for example the AG metric is not an absolute determinant of exercise difficulty, but that they are in relation to student proficiency, their background and class context. Again, we don't see this as a demerit of these indicators. The metrics created offer a quantitative measure that can aid teachers in making informed decisions about instructional planning and intervention strategies. By including both summer courses and mandatory undergraduate disciplines, our study aimed to capture insights and observations from diverse teaching and learning environments. This approach allowed us to assess the effectiveness and applicability of our plugin across different course types, catering to both general and

specialized audiences.

Future improvements for the plugin could include allowing teachers to define their own metrics based on extracted data from student submissions, providing customization options to meet specific educational needs and goals. We also will incorporate the analysis of data from other code exercise plugins like VPL. Our flexible architecture will should allow rapid iterations of the plugin and allow future work to address all aspects of the teacher experience. Overall, the integration of the PETA plugin within the Moodle LMS offers a comprehensive solution for managing and analyzing student performance in coding exercises. By leveraging metrics and visualizations, teachers can gain valuable insights, improve instructional design, and enhance the learning experience for students.

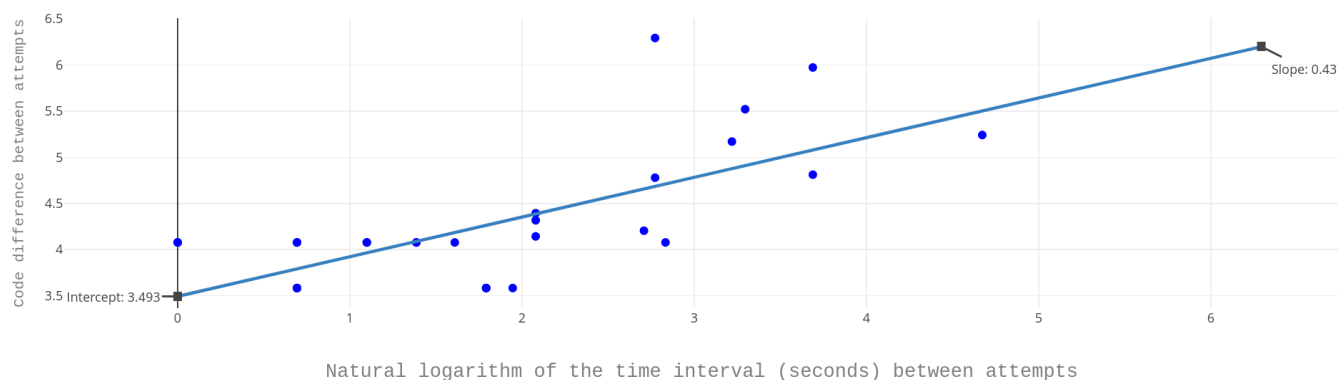


Fig. 8. Scatter plot generated by the PETA plugin for the exercise with the highest AG (Average Grade)

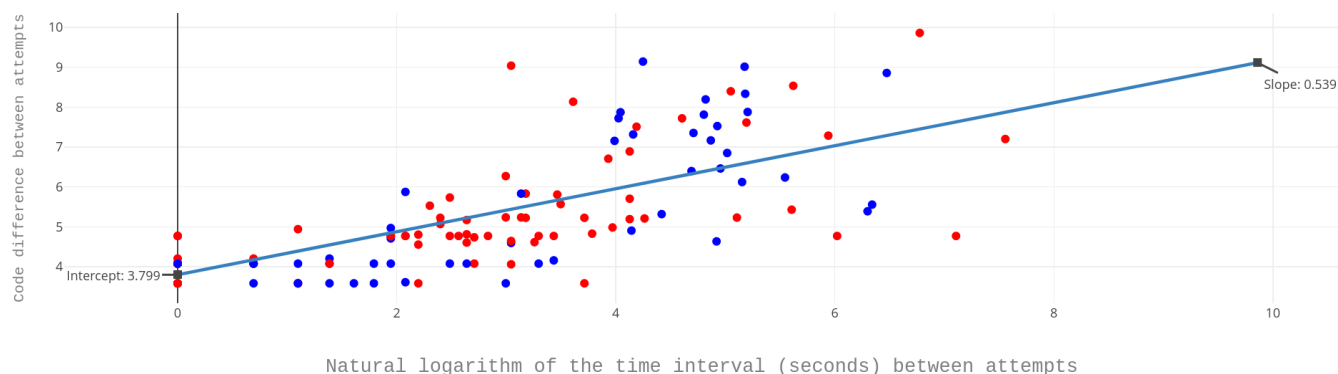


Fig. 9. scatter plot generated by the PETA plugin for the exercise with the smallest AG (Average Grade)

ACKNOWLEDGMENT

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

REFERENCES

- [1] H. Keuning, J. Jeuring, and B. Heeren, "A systematic literature review of automated feedback generation for programming exercises," *ACM Transactions on Computing Education*, vol. 19, no. 1, pp. 3:1–3:43, 2018. [Online]. Available: <https://doi.org/10.1145/3231711>
- [2] G. Biesta, "Risking ourselves in education: Qualification, socialization, and subjectification revisited," *Educational Theory*, vol. 70, no. 1, pp. 89–104, 2020, eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1111/edth.12411>. [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1111/edth.12411>
- [3] S. Papert, *Mindstorms: children, computers, and powerful ideas*. Basic Books, Inc., 1980.
- [4] A. M. Garber, "Data science: What the educated citizen needs to know," *Harvard Data Science Review*.
- [5] J. Bennesden and M. E. Caspersen, "Failure rates in introductory programming." [Online]. Available: <http://doi.acm.org/10.1145/1272848.1272879>
- [6] —, "Failure rates in introductory programming: 12 years later," *ACM Inroads*.
- [7] L. M. d. Souza, I. M. Felix, B. M. Ferreira, A. A. F. Brandão, and L. d. O. Brandão, "I know what you coded last summer," in *Anais do Simpósio Brasileiro de Informática na Educação*. SBC, 2021, pp. 909–920, ISSN: 0000-0000. [Online]. Available: <https://sol.sbc.org.br/index.php/sbie/article/view/18117>
- [8] R. E. Bennett, "Formative assessment: a critical review," *Assessment in Education: Principles, Policy & Practice*, vol. 18, no. 1, pp. 5–25, 2011-02. [Online]. Available: <http://www.tandfonline.com/doi/abs/10.1080/0969594X.2010.513678>
- [9] P. A. Rodrigues, L. d. O. Brandão, and A. A. F. Brandão, "Interactive assignment: A moodle component to enrich the learning process," in *2010 IEEE Frontiers in Education Conference (FIE)*, 2010-10, pp. T4F-1–T4F-6, ISSN: 2377-634X.
- [10] J. C. Rodríguez-del Pino, E. Rubio Royo, and Z. Hernández Figueroa, "A virtual programming lab for moodle with automatic assessment and anti-plagiarism features," in *Proceedings of The 2012 Internacional Conference on e-Learning, e-Business, Enterprise Information Systems, & e-Government*, 2012, accepted: 2013-03-12T11:24:33Z Publication Title: Proceedings of The 2012 Internacional Conference on e-Learning, e-Business, Enterprise Information Systems, & e-Government. ISBN: 1-60132-209-7. [Online]. Available: <https://accedacris.ulpgc.es/jspui/handle/10553/9773>
- [11] D. L. Dalmon, L. d. O. Brandão, A. A. F. Brandao, and S. Isotani, "A domain engineering for interactive learning modules," *Journal of Research and Practice in IT*, vol. 44, no. 3, pp. 309–330, 2012, publisher: Australian Computer Society.
- [12] L. d. O. Brandão, R. da Silva Ribeiro, and A. A. F. Brandão, "A system to help teaching and learning algorithms," in *2012 Frontiers in Education Conference Proceedings*, 2012-10, pp. 1–6, ISSN: 2377-634X.
- [13] L. d. O. Brandão, I. M. Félix, P. A. Pereira, and A. A. F. Brandão, "Evolving technology to better support teaching introductory programming inside moodle," in *2018 XIII Latin American Conference on Learning Technologies (LACLO)*, 2018, pp. 1–8.
- [14] I. M. Félix, L. M. d. Souza, B. M. Ferreira, and L. d. O. Brandão, "A study to build a new visual programming system: Fixed or contextual menu?" in *2019 Frontiers in Education Conference Proceedings*, 2019, pp. 1–8.
- [15] D. C. Briggs, *Historical and Conceptual Foundations of Measurement*

in the Human Sciences: *Credos and Controversies*, 1st ed. Routledge, 2021-09-14. [Online]. Available: <https://www.taylorfrancis.com/books/9780429275326>

- [16] S. M. Brookhart and J. H. McMillan, *Classroom assessment and educational measurement*, S. M. Brookhart and J. H. McMillan, Eds. Routledge, Taylor & Francis Group, 2020.
- [17] D. P. Goodman and R. K. Hambleton, "Student test score reports and interpretive guides: Review of current practices and suggestions for future research," *Applied Measurement in Education*, vol. 17, no. 2, pp. 145–220, 2004-07-01, publisher: Routledge_eprint: https://doi.org/10.1207/s15324818ame1702_3.
- [18] R. K. Hambleton, H. Swaminathan, and H. J. Rogers, *Fundamentals of item response theory*. Sage Publications, 1991.
- [19] M. Wilson, *Constructing measures: an item response modeling approach*. Lawrence Erlbaum Associates, 2005.
- [20] A. E. R. Association, A. P. Association, N. C. on Measurement in Education *et al.*, "Standards for educational and psychological testing," (No Title), 1985.
- [21] F. Lord, "A theory of test scores," *Psychometric monographs*, 1952.
- [22] F. M. Lord and M. R. Novick, *Statistical theories of mental test scores*. IAP, 2008.
- [23] G. Rasch, *Probabilistic models for some intelligence and attainment tests*. ERIC, 1993.
- [24] Moodle Docs. (2023) Quiz statistics report - MoodleDocs. [Online]. Available: https://docs.moodle.org/en/Quiz_statistics_report
- [25] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," in *Soviet physics doklady*, vol. 10. Soviet Union, 1966, pp. 707–710, issue: 8.
- [26] M. Invernizzi, F. Petrolí, D. Piccolotto, H. Treméa, J. Studer, M. A. Canossa, and J. M. C. d. Silva, "Uma revisão sistêmica sobre os plug-ins moodle para apoio as tarefas docente," *Anais do Computer on the Beach*, pp. 514–517, 2016. [Online]. Available: <https://periodicos.univali.br/index.php/acotb/article/view/10783>
- [27] K. Yacef, "Intelligent teaching assistant systems," in *International Conference on Computers in Education, 2002. Proceedings.*, 2002-12, pp. 136–140 vol.1.
- [28] M. A. P. Lima, L. S. G. Carvalho, E. H. T. Oliveira, D. B. F. Oliveira, and F. D. Pereira, "Uso de atributos de código para classificar a dificuldade de questões de programação em juízes online," *Revista Brasileira de Informática na Educação*, vol. 29, pp. 1137–1157, 2021-09-01. [Online]. Available: <https://sol.sbc.org.br/journals/index.php/rbie/article/view/2787>
- [29] T. Effenberger, J. Čechák, and R. Pelánek, "Measuring difficulty of introductory programming tasks," in *Proceedings of the Sixth (2019) ACM Conference on Learning @ Scale*, ser. L@S '19. Association for Computing Machinery, 2019-06-24, pp. 1–4. [Online]. Available: <https://doi.org/10.1145/3330430.3333641>