Evaluation of an intervention on activity planning in CS1

Alberto Gómez, Maria-Jesús Marco-Galindo, Julià Minguillón

***Abstract*—A key factor in online learning is an instructional design that ensures that students maintain an adequate and constant learning pace throughout the course. This is especially relevant when a fundamentally practical and progressive learning approach is required, such as in introductory programming courses. This article describes an intervention conducted in a first-year subject of the Computer Engineering degree called "Programming Fundamentals". This subject poses many challenges related to the introduction of abstract concepts, the completion of programming exercises in a specific language, and the monitoring of the pace of proposed learning activities so that students can achieve adequate learning. Based on academic results from several semesters, it was decided to make an intervention that modified the planning of learning activities to maintain motivation and learning pace throughout the semester, while reducing the time between completing the activities and receiving feedback. An analysis of the results following the change shows that more students complete the core activities, with a decrease in dropouts from continuous assessment and an increase in the number of students passing the course. Data analysis has been validated using *propensity score matching*, a method for evaluating interventions with a quasi-experimental design.**

***Index Terms*— CS1, activity planning, formative assessment, intervention evaluation, performance analysis, Propensity Score Matching.**

I. Introduction

EXPERIENCE demonstrates that learning to program is difficult, and teaching programming is a challenge, as evidenced by the extensive number of publications on the subject, as described, for example, in [1], which analyzes 1666 studies on programming learning. Therefore, the literature reflects a widespread agreement that learning to program is a challenging process for most students at all levels. Particularly, the dropout rate in introductory programming courses at the university level is generally high, while success rates are low [2]. Despite the extensive literature on the topic and decades of experience and research, many questions remain unanswered. Why is learning to program so difficult for many students? What factors determine a student's success in an introductory programming course? It appears that no definitive factor predicts success [3], but it is clear that one of the elements that positively influence learning effectiveness is practical programming activities. There is a necessary and indispensable combination of fundamental knowledge (knowing) and practical knowledge (knowing how to apply). The abstract aspects of algorithmic thinking must be put into practice by coding in a specific programming language through a well-designed laboratory activity strategy, allowing students to work on them individually at first and then progressively integrate them into larger projects [2].

From the student's perspective, many studies analyze which content areas are more challenging and the cognitive load they entail [4], as well as the most suitable teaching and learning strategies to reduce such load [5], including individualized feedback, which is crucial to receive at critical moments when it can be most effective [6]. One of these critical moments is the initial weeks of the course, during which close attention should be paid to ensure that each student has a positive initial experience, facilitating their learning and providing quick assistance to those who show signs of disengagement, such as not submitting the first assignment or not actively participating in class [7]. Related to the aforementioned, it is also important to study factors that positively influence students' engagement in the subject, encouraging them to start working on activities from the beginning and committing to their learning without losing interest or decreasing their dedication throughout the course. In this regard, the studies by Kanaparan [8] regarding engagement in an introductory programming course are relevant, expressed in terms of the three indicators that determine it: effort, persistence, and seeking support.

Setting specific short-term goals enhances student motivation [9]. Therefore, assessment activities throughout the course maintain consistent engagement with the materials, while also serving as formative and summative evaluation [10], [11]. In online learning environments, where unfortunately dropout rates are high, the design of an activity plan should aim to maintain student motivation, effort, and participation [12].

In [13], a review of 32 studies is presented, describing interventions in introductory programming courses and assessing the most influential changes leading to improvements. These changes mainly consist of increasing

Alberto Gómez is with Universidad de Extremadura and with Universitat Oberta de Catalunya (e-mail: agomezma@uoc.edu).

Maria-Jesús Marco-Galindo is with Universitat Oberta de Catalunya (e-mail: mmarcog@uoc.edu).

Julià Minguillón is with Universitat Oberta de Catalunya (e-mail: jminguillona@uoc.edu).

DOI (Digital Object Identifier) Pendiente

collaboration among students, modifying and contextualizing content, creating prerequisite courses, adjusting the grading system, and providing more tutoring. One of the main conclusions drawn by the authors of this review is that, in general, all interventions improve results when compared to traditional approaches, although there are no statistically significant differences in effectiveness between most of them.

One of the challenges in analyzing the improvement achieved through an intervention is that it requires comparing results obtained in different semesters with different student populations. Generally, it is assumed that the characteristics of the student populations remain constant across the compared courses, although this may not necessarily be the case. In our case, we have used a quasi-experimental design to validate the comparison of data from four distinct semesters.

This article extends the research presented and published at the CINAIC 2021 conference, which was selected for submission to IEEE-RITA [14]. In comparison to the original article, we expand the literature review and the analysis of the results and present the validation of the intervention using the quasi-experimental design method known as p*ropensity score matching*.

This work is structured as follows: Section 2 describes the quasi-experimental design. Next, Section 3 provides a detailed description of the methodology employed, including the context, data used, and the measure used to evaluate the intervention. The subsequent section describes the initial situation and the intervention. Section 5 presents the main results obtained, along with their validation. The final section describes the main conclusions and outlines future research directions.

II. Intervention Analysis

To evaluate the effect of an intervention in a course, it is not sufficient to compare outcome indicators obtained by students in the semesters before and after the change, as these students belong to different populations. Directly comparing data from different semesters carries the risk of significant differences in student profiles in each semester, thus biasing the evaluation results due to significant differences in the datasets, not just the influence of the intervention itself.

A. *Quasi-experimental Design*

To directly compare results from multiple semesters, a randomized experiment should be conducted on the same population, with a control group that follows the previous curriculum and another group with the new activity distribution, randomly assigned each semester.

In fields such as medicine, experimental design must always be randomized, where the control and experimental groups are formed following a strict methodology that ensures other factors do not influence the study, maintaining its validity [15]. In educational research, it is challenging and, in some cases, unethical to conduct randomized studies with control groups [16].

For cases where randomized trials cannot be conducted, quasi-experimental statistical techniques attempt to control, as much as possible, the influence of other variables besides the one being analyzed [17]. Quasi-experimental designs involve selecting similar elements from both data groups (the group experiencing the intervention and the group that does not) so that the two final samples being compared can be considered equivalent and similar to what would be obtained with a randomized experimental design. These methods aim to reduce potential bias caused by other confounding variables that affect individuals' assignment to one group or the other. When the assignment to groups is random, the effect of confounding factors is balanced, and bias becomes insignificant.

B. *Propensity Score Matching*

Among the most commonly used quasi-experimental techniques is the Propensity Score Matching (PSM) method [18], [19]. The propensity score (PS) of an observation is the probability or propensity of being part of the group that underwent the intervention, calculated based on the explanatory variables available. If two observations have the same values for the considered covariates, it is assumed that the probability of their participation in the intervention would be the same. By selecting one observation from the control group and another from the group that underwent the intervention, a similar result is obtained as if the observations had been randomly assigned.

Typically, PS is calculated using a logistic regression model, with participation or non-participation in the intervention as the outcome and some variables in the model as predictors. Subsequently, the data is matched based on similar PS values from the two initial datasets, discarding elements that cannot be matched. Once the matched elements are obtained, the impact of the intervention can be analyzed as if the data were derived from a randomized study. The matching between the two groups can be performed according to various similarity criteria, resulting in different sets of matched data.

This method has limitations since bias reduction may be partial if there are not enough significant explanatory variables or if too many observations are lost during matching, rendering the matched data no longer representative [20]. Therefore, it is important to verify after matching that too many observations have not been lost and that the characteristics of the intervention and control groups are balanced across most of the considered variables [21].

In [19], Harris provides an extensive literature review on the use of PSM in higher education, including a case study comparing test results based on the students' enrolled programs of study. As admission requirements and program characteristics differ, entrance grades, student profiles, or motivation can significantly influence the results, beyond the specific program of study. Directly comparing results without applying PSM would disregard these differentiating factors and may lead to incorrect conclusions. In [22], PSM is used to eliminate bias when a control group could not be randomly formed, and the scope of the results is always limited by the potential confounding factors considered in the study.

Various algorithms can be used to match observations with similar propensity scores. In this work, several commonly used algorithms were employed to compare the resulting matched datasets and select those that exhibited a greater degree of balance. Several R packages assist in the PSM process (e.g., MatchIt [23] or Matching [24]), as well as in result visualization and interpretation (e.g., Cobalt [25] or PSAgraphics [26]). In this analysis, MatchIt and Cobalt were primarily utilized.

The matching algorithms used, according to the names used in the MatchIt package, are as follows:

- *Exact Matching*: Observations that have exactly the same values for all variables are matched.
- Subclassification: A predetermined number of disjoint classes is formed, where the distribution of covariates is as similar as possible.
- *Nearest Neighbor Matching*: The PS is used to match each observation from one group with the most similar value in the other group. The assignment is performed in order using a greedy algorithm, sometimes resulting in matching observations with very different values. An additional option, a caliper, can be used to ensure that the difference between matched values is smaller than a specified proportion of the standard deviation of the calculated distances. Additionally, this type of matching can be combined with exact matching for selected variables.
- *Full Matching*: As many disjoint classes as necessary are formed. Each class contains one observation from one of the two groups and as many observations from the other group as they match with the first observation.

III. Methodology

This section describes the scenario in which the proposed intervention was designed and implemented, according to the following methodology.

The main change in the intervention consisted of dividing the continuous assessment activities into shorter segments and requiring them to be submitted every week, aiming for a more continuous work process and more frequent overall feedback, following what has been seen in the literature.

A. Context

The course "Fundamentals of Programming" is a mandatory course in the Bachelor's programs of Computer Science and of Telecommunication Technologies Engineering at the Universitat Oberta de Catalunya. It is also an elective course in some specialized master's programs and part of the university's open program.

As a virtual university, it has a highly heterogeneous student body. However, all students take the course in the same manner and in the same virtual learning environment.

In summary, 82.17% of the students are male, and the median age at the time of taking the course is 30 years (with the most popular age group being 31-40 years, followed by 26-30 years). Most students simultaneously take two to three courses, a typical situation for part-time students.

The activities are accompanied by the necessary resources to complete them: theoretical content on algorithmic thinking, examples, and instructions for coding in C, as well as a virtual machine with the Codelite development environment for programming. Once the deadline for a Continuous Assessment Activity (CAA) or a programming practice (PR) has passed, the solutions are published, and each student receives a grade for their exercise. The continuous assessment activities are graded using an alphabetical system: A (very good), B (good), C+ (sufficient), C- (low), D (very low), and N (not submitted). However, the final grade is numerical, according to the Spanish grading system.

The course instructors use the virtual classroom's announcement board to communicate any issues related to the activities. Student questions, on the other hand, are shared and resolved through the classroom forum or via personal messages between the instructor and the student. The forum is a space where students are expected to participate by sharing their questions and collaboratively creating knowledge. For the practical part of the course, students have the support of the programming laboratory, where a laboratory instructor helps with programming environment issues and C code problems [27].

As an introductory programming course, it covers the basic principles of algorithmic thinking combined with programming practices in the C programming language. Continuous assessment is combined with three algorithmic design activities, two C programming practices, and a final in-person exam. The three CAAs are optional, allowing students to decide how many and which ones they want to complete, knowing that each one contributes to the final grade. The two programming practices (PR1 and PR2) and the final exam (EX) are mandatory.

B. Data Used

To analyze whether the changes introduced in this intervention have reduced dropout rates and improved academic performance, data from four semesters were utilized: two prior to the modification of the activity plan (the second semester of the 2016-17 academic year, 2016/2, and the first semester of the 2017-18 academic year, 2017/1) and two after the changes (the second semester of the 2017-18 academic year, 2017/2, and the first semester of the 2018-19 academic year, 2018/1). This allows for a comparison of an entire course before and after the intervention. The data were obtained from the institutional Learning Record Store [28], resulting in a total of 1487 observations, with 721 prior to the change and 766 after, where each observation represents a student enrolled in one of the four analyzed semesters.

Each observation describes the profile and grades of a student in a given semester. The main variables analyzed can be grouped into three categories:

- Sociodemographic profile: gender, age group.
- Study profile: enrolled degree, admission pathway, whether the student comes from vocational training or

university studies, number of courses taken in the semester, number of semesters enrolled in the degree program, and number of times the course has been repeated.
- Obtained grades: grades for each CAA, PR1, PR2, final grade for the programming practices, and final grade for continuous assessment.

C. *Evaluation Measures*

The main objective of the analysis is to determine if the implemented changes have positively influenced student performance by reducing high dropout rates in the course. The instructional design and evaluation model are comparable in both cases until the delivery of the first programming practice PR1. From that point onward, after a significant portion of the semester has elapsed, there are some changes in the consideration of PR2 and minor modifications to the syllabus.

Passing the course is primarily determined by successfully completing PR1. The delivery of the first programming practice (PR1) is one of the clearest indicators of a student's final performance in the course. Among those who submit PR1, 83.4% pass the programming practices, and 93.3% pass continuous assessment. Only 21% of those who do not submit the first programming practice pass the continuous assessment of the course (although they cannot pass the course since PR1 is a requirement).

Therefore, the analysis focuses on how the implemented changes have influenced the submission rates of PR1.

IV. Intervention Framework

This section provides a detailed description of the starting point and the fundamental changes introduced by the intervention.

A. *Starting Point*

With the activity plan described in the context, the number of students who dropped out of the course was quite high, and the final performance was low, always around 30%. These results are typical and in line with the literature [29], [30], [31], and they were already accepted as normal for an introductory programming course, especially in a fully online teaching environment. Analyzing the students' activity in detail over several semesters, it was observed that the main problem was dropout during the first few weeks of the semester. This dropout was attributed to various factors inherent to learning programming:

- Programming has a steep initial learning curve. Understanding the fundamental abstract concepts of algorithmic thinking, which are the basis of programming, can be challenging at the beginning. As a result, many students become lost from the start and, unable to solve the initial simpler activities, end up dropping the course.
- Programming learning is cumulative, so if a student fails to understand an initial concept and complete the first activities, it becomes difficult for them to continue with subsequent concepts and activities.
- In an online learning environment, learning is structured through continuous assessment activities proposed to students throughout the semester. These activities determine the pace of work for students through study and practice. Additionally, feedback received from these activities provides guidance for overcoming difficulties and progressing further. If the pace is not maintained, it is easy to give up.

Inspired by the success of previous changes made in mathematics courses within the program, which managed to reverse this trend [32], an intervention was planned for the 2017-2018 academic year to smoothen the learning curve and establish a more continuous work rhythm. The main goal was to reduce dropout rates and increase the number of students who could successfully complete the course.

B. *Description of the Intervention*

In this intervention, there were no changes to the course objectives, teaching resources, syllabus, or expected learning outcomes. The teaching staff also remained the same.

The redesign of the instructional model focused on making substantial changes to the number and format of the continuous assessment activities proposed to students throughout the semester.

The aim was to establish a more continuous work rhythm for students, increase their motivation and desire to program, and help them become accustomed to the work dynamics from the beginning of the semester. Based on the principle that programming is learned by programming, the continuous assessment activities were reinforced and restructured in several aspects:

- Shorter activities, each focusing on a single fundamental concept.
- More frequent activities, with one activity per week.
- All activities, both the Continuous Assessment Activities (CAAs) and the programming practices (PRs), are linked together through a common context (a practical case).
- Each student receives feedback on every activity.

The change in the activity plan mainly involved dividing each previous CAA into multiple shorter CAAs to increase the frequency of submissions. The type of problems remained exactly the same; only the exercises were reorganized into three blocks with multiple submissions. CAA1 was divided into 4 (the new CAA1 to CAA4), CAA2 into 4 (the new CAA5 to CAA8), and CAA3 was split into 2 (the new CAA9 and CAA10).

With this new plan, submissions became weekly, and students received feedback every week with the publication of the solution and general comments for the whole group. Individual evaluation and grading by the instructor were maintained at three points during the semester, after each block, i.e., right after the fourth, eighth, and tenth activities (CAA4, CAA8, and CAA10) respectively.

The new continuous assessment model consists of a sequence of short, weekly activities, each containing a single exercise that combines theory (algorithmic design) and practice (C programming). Specifically, a total of ten continuous assessment activities are proposed, which are combined with the completion of two mandatory comprehensive programming exercises that integrate all course content (PR1 and PR2). More specifically, the sequence of activities is as follows:

- CAA1: Exercise to work on basic data types.
- CAA2: Exercise on expressions with basic types.
- CAA3: Exercise to practice the conditional structure.
- CAA4: Exercise on the use of iterative structures.
- CAA5: Exercise requiring the use of tuples.
- CAA6: Exercise on actions and functions (modularity).
- CAA7: Exercise requiring parameter passing.
- CAA8: Exercise requiring the use of tables.
- PR1: Integration practice that encompasses all learned content in a more extensive C programming project.
- CAA9: Exercise introducing a simple abstract data type: stack, queue, or list.
- CAA10: Exercise to practice operations with stacks, queues, and lists.
- PR2: Adding an abstract data type structure to the PR1 project.

Both the two programming exercises (PRs) and the final in-person exam (EX) remained unchanged.

The weekly tests or activities are aligned with the programming practices and are part of the same context (practical case), so the weekly programming exercises are later used in part of the programming practices. This simplifies the understanding of the context of the programming practices and facilitates their resolution. Although continuous assessment is optional, these programming practices are mandatory, and it is also necessary to pass them to pass the course. Table I summarizes the evaluation model.

TABLE I
EVALUATION MODEL

	Number	Mandatory	In-person/Virtual
CAAs	3 (before) / 10 (after)	No	Virtual
PRs	2	Yes	Virtual
Exam	1	Yes	In-person

In terms of the continuous assessment grade, the CAAs account for 40% and the PRs for 60%. This grade is combined with the exam grade according to the following percentages: 70% continuous assessment grade and 30% exam grade. If the CAAs are not completed, the final grade is calculated according to the following formula: 40% for the practical grade and 60% for the exam grade.

V. Results

The intervention shows a clear improvement in the percentage of students who submit the first programming practice (PR1). The percentage of students who submit PR1 has increased in the semesters following the intervention, from 48.13% to 60.70%.

The intervention also shows a significant increase in the percentage of students who pass the course and a noticeable decrease in the percentage of non-submissions in continuous assessment.

Figures 1 and 2 illustrate the evolution of student grades in the different continuous assessment activities leading up to the submission of PR1 before and after the intervention. The two colors indicate whether the student submitted PR1 or not (green and brown, respectively). It can be observed that the number of dropouts (non-submissions, N) increases as the semester progresses. With the original activity plan, there is a large group of students who do not submit the first CAA and do not submit any subsequent activities. In the case of the new plan, this number is lower, and it can also be observed that some students who initially receive low grades in the early submissions manage to catch up and pass the remaining CAAs and submit the programming practice.

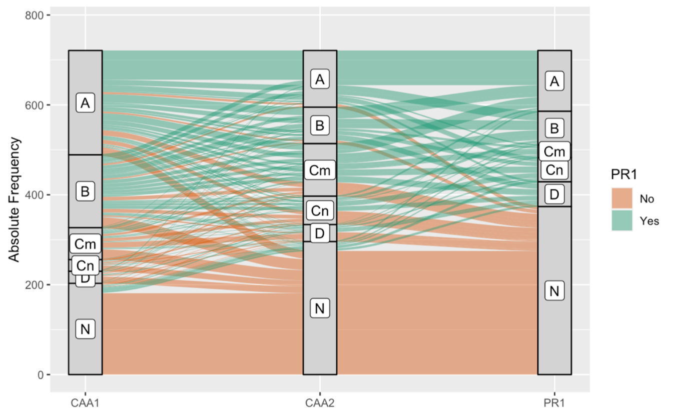

Figure 1. Relationship between programming practice grades and continuous assessment if PR1 is presented in previous semesters before the intervention.

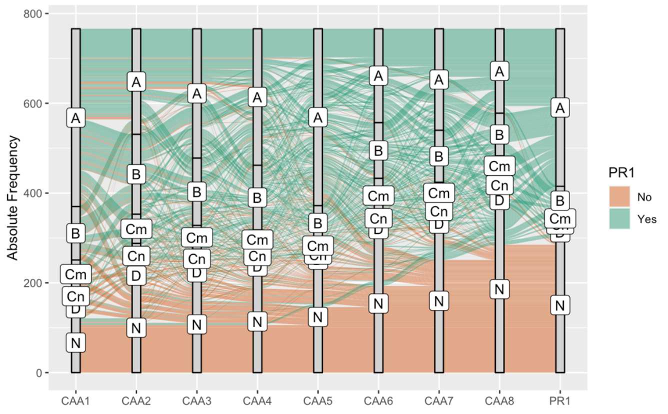

Figure 2. Relationship between programming practice grades and continuous assessment if PR1 is presented in subsequent semesters after the intervention.

Both graphs also show the general trend of maintaining grades at similar levels from one activity to the next: students who start with grades of A or B tend to stay in the same range. However, students who start with poor results tend to drop out at a high rate.

It can be observed that the number of students who do not submit CCA1 before the change is twice the number of those who do not submit CCA4 after the intervention. These submissions correspond to the same week of the course. There is still a difference, although smaller, when comparing the submission of CCA2 before with CCA8 in the subsequent semesters.

TABLE II
DISTRIBUTION OF GRADES - PR1

	A	B	C+	C-	D	N
Before	9 %	15 %	8 %	2 %	12 %	54 %
After	27 %	23 %	6 %	2 %	3 %	39 %

Table II shows the breakdown of grades obtained by students before and after the intervention. An increase in passing grades can be observed, especially in the Good (B) and Excellent (A) categories. Moreover, there is a significant decrease in non-submissions (N). The change in the activity plan also seems to have improved the final grades of the programming practices, increasing the number of passing grades.

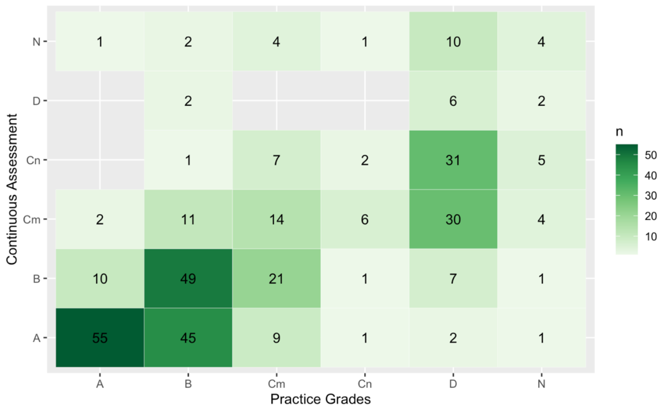

Figure 3. Relationship between programming practice grades and continuous assessment if PR1 is presented in the semesters prior to the intervention.

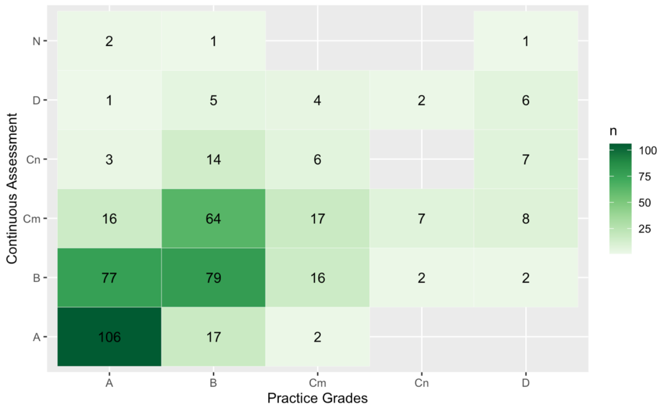

Figure 4. Relationship between programming practice grades and continuous assessment if PR1 is presented in the semesters following the intervention.

Figures 3 and 4 present two heatmaps derived from the data before and after the intervention, respectively, relating the grades of the practices and the continuous assessment if PR1 was submitted. The high correlation between passing or failing both components can be observed in both cases.

However, in the semesters prior to the changes in the activity plan, there is a significant group of students who pass the continuous assessment with low grades (C+, C-) but fail the programming practices (D). In the semesters following the intervention, this group is much smaller.

Based on the results and the interpretation, it is possible to consider the implications that this analysis may have for the future of the course. On one hand, it seems reasonable to think that increasing the number of activities smooths the initial learning curve and establishes a more continuous work rhythm, encouraging student participation in the course from the beginning. These two factors increase the likelihood of passing the course and also lead to better results. As shown in Table III, the course performance significantly improved in the semesters following the intervention.

TABLE III
EVOLUTION OF PERFORMANCE AND FOLLOW-UP OF CONTINUOUS ASSESSMENT

	2016/2	2017/1	2017/2	2018/1
Performance	34.3 %	23.5 %	52.8 %	48.72 %
Follow-up	60.2 %	61.9 %	86.2 %	87.0 %

The perception collected through a survey conducted during the first semester of the intervention also largely indicates a positive evaluation by students. They argue that learning gradually but consistently is better than doing it all at once in an intermittent manner. Some students explain that at first, seeing so many scheduled activities intimidated them, and this schedule even caused them stress. However, as they started submitting the first few activities, they became accustomed to the dynamics and realized the benefits. In the new proposal, they also see areas for improvement, such as receiving the grade and feedback for each activity earlier, without having to wait for each block to finish.

However, despite the positive results, it is important to consider that the intervention has also involved a significant increase in workload for the instructors, especially due to the additional time required for corrections and the complexity of preparing interconnected continuous assessment activities related to the same context as the programming practices. Additionally, for some students, keeping up with a weekly activity rhythm is challenging, especially if they are simultaneously taking other courses with a similar approach. Although students did not perceive an increased workload but rather a more gradual distribution throughout the semester, it would be interesting to assess the impact of this intervention on other concurrent courses. Since the 2021-2022 academic year, the assessment load has been softened by automating the correction of some activities.

A. Validation of the Analysis with PSM

In order for these pre- and post-intervention indicators to be truly comparable, we need to ensure that the populations in both periods are similar.

PSM (Propensity Score Matching) analysis has been used to match the pre- and post-intervention data. After matching the data, it was checked whether a significant amount of data was discarded to determine if a sufficiently representative subset of the original data was retained. Additionally, it was verified that the distribution of the variables considered remained balanced in both sets.

Tests were performed using different matching algorithms (Exact Matching, Subclassification, Nearest Neighbor Matching with and without caliper, and Full Matching), although the results were not valid for some of them. For example, with the Exact Matching algorithm, the perfect balance was achieved in all variables, but over half of the observations were lost because only observations in each set that had identical values in all variables were matched. In other matching algorithms, students from groups that were underrepresented, such as women or students from minority study programs, were discarded, as was the case with the Nearest Neighbor Matching algorithm without a caliper. In these cases, the matched dataset became unbalanced and lost generality.

The results were satisfactory with few data losses (only 7 cases discarded) and balanced final variables in the Nearest Neighbor Matching algorithms with calipers of 0.1 and 0.2 (the most used values), as well as with the Full Matching algorithm, which provided the best results. These are the most commonly used matching algorithms and are described in the literature on PSM [21].

The Full Matching algorithm returned a larger set of matched data without imbalances in the covariates. A detailed analysis of the results shows that only 7 observations were discarded, and balance is adequate in all covariates. Figure 5 shows that the distribution of matched PS (Propensity Scores) in the pre- and post-intervention datasets using the Full Matching algorithm is similar. The unmatched observations (at the top of the figure) correspond to cases whose propensity scores fall outside the common range.

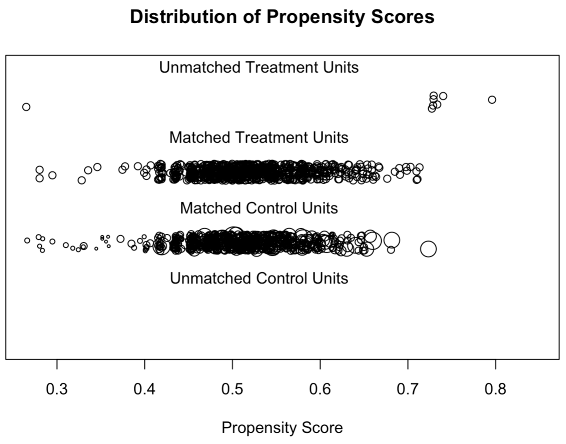

Figure 5. Distribution of PS with Full Matching algorithm.

Figure 6 allows a comparison of the histograms of the propensity scores. The distributions are very similar before and after matching (only a few elements were discarded), as well as in the control group and the post-intervention group. Therefore, the matching process appears to be appropriate because it retains almost all the original data, and there are no noticeable differences in the characteristics of the pre-intervention and post-intervention student populations, as demonstrated by the covariate balance analysis.

All the considered covariates are balanced after matching, as can be seen in Figure 7, which compares the balance before and after matching of sociodemographic and educational profile covariates. A detailed analysis of the data verifies that all covariates and their interactions remain balanced.

With fairly balanced results and a low number of discarded observations in several matching algorithms, it can be inferred that the student profiles in the pre-intervention and post-intervention datasets were quite similar. Therefore, the analysis conducted in the previous section on performance and follow-up is appropriate. Using this data, several logistic regression models were presented and analyzed in [33] to quantify the influence of the intervention on improving the rate of submissions of the first programming practice.

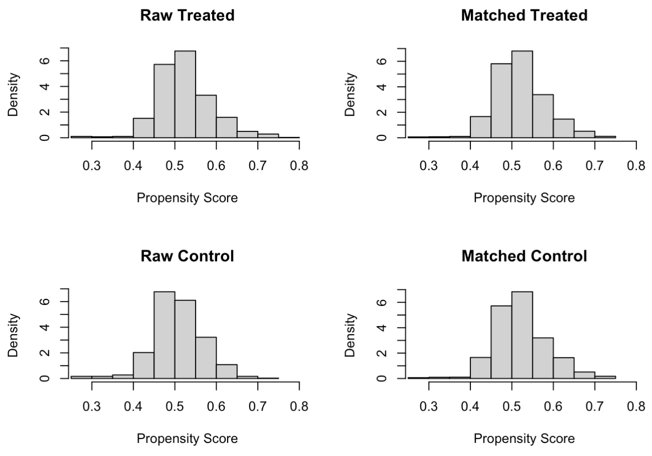

Figure 6. Histogram of PS with Full Matching algorithm.

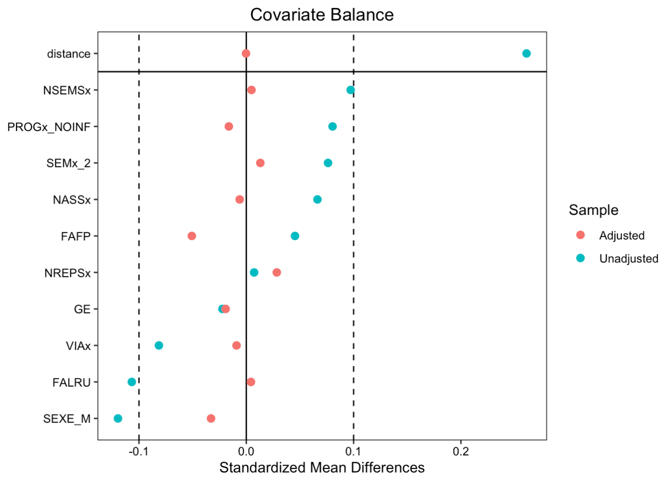

Figure 7. Covariate balance with Full Matching algorithm.

VI. Conclusions

In summary, what is most relevant for completing the mandatory activity (and consequently passing the course) is that students engage with the course from the beginning by completing the continuous assessment activities. In this way, the most effective support occurs in the early weeks of the course: on one hand, by providing feedback as soon as possible to help and encourage students to progress to the next activities, and on the other hand, by identifying students who do not submit the initial activities and offering them the support they need to get started as soon as possible.

This improvement is aligned with the findings presented in the research literature on computer science education. Increasing feedback improves results and reduces student attrition. With more frequent activity submissions, students receive group feedback more often through general comments from the instructor and the published solution for each activity. This way, students who submit activities on time receive information about their progress every week and can seek help if needed. The change in the activity schedule also helps students maintain a more continuous, constant, and progressive work rhythm, which smooths the learning curve and improves the final performance. As a trade-off, it requires better time management because there are submissions every week.

Based on the results of this research, future work involves designing, implementing, and evaluating another intervention that enhances more individualized feedback and support for students in the initial activities, with the aim of recovering students at risk of dropping out or failing the course as soon as possible.

Finally, from a methodological perspective, it is important to use quasi-experimental techniques like PSM to correct for potential biases resulting from the comparison of potentially different populations.

Acknowledgment

The authors would like to thank the Editorial Committee of CINAIC 2021 for inviting them to publish the article in the VAEP-RITA journal. This research was partially funded by project 2021SGR01412 STEAM University Learning Research Group from the Generalitat de Catalunya.

References

[1] A. Luxton-Reilly *et al.*, "Introductory programming: A systematic literature review," *Annu. Conf. Innov. Technol. Comput. Sci. Educ. ITiCSE*, pp. 55–106, Jul. 2018, doi: 10.1145/3293881.3295779.

[2] C. Watson and F. W. B. Li, "Failure Rates in Introductory Programming Revisited," in *Proceedings of the 2014 Conference on Innovation in Computer Science Education*, in ITiCSE '14. New York, NY, USA: Association for Computing Machinery, 2014, pp. 39–44. doi: 10.1145/2591708.2591749.

[3] A. S. Carter, C. D. Hundhausen, and O. Adesope, "Blending measures of programming and social behavior into predictive models of student achievement in early computing courses," *ACM Trans. Comput. Educ. TOCE*, vol. 17, no. 3, pp. 1–20, 2017.

[4] J. Sorva, *Visual program simulation in introductory programming education*. Aalto University, 2012.

[5] R. Hoda and P. Andreae, "It's not them, it's us! Why computer science fails to impress many first years," in *Proceedings of the Sixteenth Australasian Computing Education Conference-Volume 148*, 2014, pp. 159–162.

[6] C. Ott, A. Robins, and K. Shephard, "Translating principles of effective feedback for students into the CS1 context," *ACM Trans. Comput. Educ. TOCE*, vol. 16, no. 1, pp. 1–27, 2016.

[7] L. Porter and D. Zingaro, "Importance of early performance in CS1: two conflicting assessment stories," in *Proceedings of the 45th ACM technical symposium on Computer science education*, 2014.

[8] G. Kanaparan, R. Cullen, and D. D. Mason, "Self-Efficacy and Engagement as Predictors of Student Programming Performance.," in *PACIS*, 2013, p. 282.

[9] D. Bueno, *Neurociencia para educadores*, 3rd ed. Octaedro, 2017.

[10] J. Biggs and C. Tang, *Teaching for Quality Learning at University*, 4th ed. SRHE and Open University Press, 2003.

[11] B. E. Vaessen, A. van den Beemt, G. van de Watering, L. W. van Meeuwen, L. Lemmens, and P. den Brok, "Students' perception of frequent assessments and its relation to motivation and grades in a statistics course: a pilot study," *Assess. Eval. High. Educ.*, vol. 42, no. 6, pp. 872–886, 2017, doi: 10.1080/02602938.2016.1204532.

[12] A. Sangra, "Decálogo para la mejora de la docencia online: propuestas para educar en contextos presenciales discontinuos," *Decál. Para Mejora Docencia Online*, pp. 1–215, 2020.

[13] A. Vihavainen, J. Airaksinen, and C. Watson, "A Systematic Review of Approaches for Teaching Introductory Programming and Their Influence on Success," in *Proceedings of the 10th Annual Conference on International Computing Education Research*, in ICER '14. New York, NY, USA: ACM, 2014, pp. 19–26. doi: 10.1145/2632320.2632349.

[14] M.-J. Marco-Galindo and J. Minguillón, "La evaluación formativa como factor decisivo en el aprendizaje online. Intervención en una asignatura inicial de programación," in *Actas del VI Congreso Internacional sobre Aprendizaje, Innovación y Cooperación (CINAIC 2021)*, 2021, pp. 677–681.

[15] G. Cousin, *Researching Learning in Higher Education: An Introduction to Contemporary Methods and Approaches*. in SEDA Series. Taylor & Francis, 2009. [Online]. Available: https://books.google.es/books?id=I3WQAgAAQBAJ

[16] A. J. Ko and S. A. Fincher, "A Study Design Process," in *The Cambridge Handbook of Computing Education Research*, Cambridge University Press, 2019, pp. 81–101. doi: 10.1017/9781108654555.005.

[17] M. M. Arias, "Lectura crítica en pequeñas dosis Índices de propensión. El deseo de parecerse al ensayo clínico," *Rev Pediatr Aten Primaria*, vol. 17, pp. 87–90, 2015.

[18] C. Ramirez Ovalle, "Sobre la Técnica de Puntajes de Propensión (Propensity Score Matching) y sus usos en la investigación en Educación," *Educ. Cienc.*, vol. 4, no. 43, pp. 81–89, 2015.

[19] H. D. Harris, "Propensity score matching in higher education assessment," *Masters Theses*, May 2015, [Online]. Available: https://commons.lib.jmu.edu/master201019/55

[20] S. Guo, M. Fraser, and Q. Chen, "Propensity score analysis: Recent debate and discussion," *J. Soc. Soc. Work Res.*, vol. 11, no. 3, pp. 463–482, Sep. 2020, doi: 10.1086/711393/ASSET/IMAGES/LARGE/FG1.JPEG.

[21] H. Harris and S. J. Horst, "A brief guide to decisions at each step of the propensity score matching process," *Pract. Assess. Res. Eval.*, vol. 21, no. 4, 2016.

[22] L. A. Lim *et al.*, "What changes, and for whom? A study of the impact of learning analytics-based process feedback in a large course," *Learn. Instr.*, 2019, doi: 10.1016/j.learninstruc.2019.04.003.

[23] D. E. Ho, K. Imai, G. King, and E. A. Stuart, "MatchIt: Nonparametric Preprocessing for Parametric Causal Inference," *J. Stat. Softw.*, vol. 42, no. 8, pp. 1–28, 2011.

[24] J. S. Sekhon, "Multivariate and Propensity Score Matching The Matching package for R," *J. Stat. Softw.*, vol. 10, no. 2, 2008.

[25] N. Greifer, *cobalt: Covariate Balance Tables and Plots*. 2020. [Online]. Available: https://cran.r-project.org/package=cobalt

[26] J. E. Helmreich and R. M. Pruzek, "PSAgraphics: An R package to support propensity score analysis," *J. Stat. Softw.*, vol. 29, no. 6, pp. 1–23, 2009, doi: 10.18637/jss.v029.i06.

[27] M. J. Marco Galindo and J. Prieto Blázquez, "Necesidades específicas para la docencias de programación en un entorno virtual," *Actas JENUI*, pp. 5–12, 2002.

[28] J. Minguillón, J. Conesa, M. E. Rodríguez, and F. Santanach, "Learning Analytics in Practice: Providing E-Learning Researchers and Practitioners with Activity Data," in *Frontiers of Cyberlearning*, Springer, 2018, pp. 145–167.

[29] A. V. Robins, "Novice Programmers and Introductory Programming," in *The Cambridge Handbook of Computing Education Research*, S. A. Fincher and A. V. Robins, Eds., in Cambridge Handbooks in Psychology. Cambridge University Press, 2019, pp. 327–376. doi: 10.1017/9781108654555.013.

[30] A. Luxton-Reilly, "Learning to Program is Easy," in *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*, New York, NY, USA: Association for Computing Machinery, 2016, pp. 284–289. doi: 10.1145/2899415.2899432.

[31] G. Bain and I. Barnes, "Why Is programming so hard to learn?," in *ITICSE 2014 - Proceedings of the 2014 Innovation and Technology in Computer Science Education Conference*, Association for Computing Machinery, 2014, p. 356. doi: 10.1145/2591708.2602675.

[32] T. Sancho-Vinuesa, R. Masià, M. Fuertes-Alpiste, and N. Molas-Castells, "Exploring the effectiveness of continuous activity with automatic feedback in online calculus," *Comput. Appl. Eng. Educ.*, vol. 26, no. 1, pp. 62–74, 2018.

[33] A. Gómez, M. J. Marco-Galindo, J. Minguillón, and J. Escayola-Mansilla, "Análisis de la mejora de los resultados de una asignatura inicial de programación tras cambiar la planificación de actividades," in *Actas de las Jornadas sobre Enseñanza Universitaria de la Informática (JENUI)*, 2021, pp. 83–90.

Alberto Gómez holds a Bachelor's degree in Computer Science from the Universitat Politècnica de Catalunya (UPC). He obtained a Master's degree in Data Science from the Universitat Oberta de Catalunya (UOC). As a professor at the University of Extremadura, he teaches programming courses primarily. He has participated in and led various educational innovation projects related to new methodologies, learning analytics, and teaching programming. He is a member of AENUI (Association of University Teachers of Computer Science), where he received the 2019 Award for Teaching Quality and Innovation.

Maria-Jesús Marco Galindo holds a Bachelor's degree in Computer Science from the Universitat Politècnica de Catalunya (UPC). She earned a Ph.D. in Education and ICT from the Universitat Oberta de Catalunya (UOC). Since 1999, she has been a professor at UOC, where she conducts research within the STEAM University Learning Research Group (EduSTEAM), focusing on teaching programming and transversal competencies in virtual environments. She is a member of AENUI (Association of University Teachers of Computer Science), where she received the 2023 Award for Teaching Quality and Innovation.

Julià Minguillón holds a Ph.D. in Computer Engineering from the Universitat Autònoma de Barcelona (UAB). Since 2001, he has been a professor at UOC, where he conducts research within the EduSTEAM group on topics related to student analysis in virtual environments, learning analytics, and the use of interactive data visualizations to support teachers and students.