Global Journal of Engineering Education

Using an interactive software tool for the formative and summative evaluation in a computer programming course: an experience report

Felipe Restrepo-Calle, Jhon J. Ramírez-Echeverry & Fabio A. González

Universidad Nacional de Colombia
Bogotá, Colombia

ABSTRACT: This article describes the use of an automatic tool that supports both formative and summative evaluation, in an introductory computer programming course. The aim was to test the tool in a real scenario and to assess its potential benefits. The participants were 56 students in an engineering school. Different programming tasks were solved by students with the support of the tool. It provides summative evaluation using automatic grading, and formative evaluation through real-time checking of good coding practices, code execution visualisation, testing code with custom inputs, informative feedback on failed test cases, and student performance statistics for self-monitoring the learning process. Results of the experience show the students considerably used the tool, denoting high engagement to achieve the learning goals of the tasks. The students perceived the tool was highly useful to debug programs and carry out frequent practice of programming, thanks to the automatic feedback and test of solutions for programming assignments. Results offer evidence of the importance of providing comprehensive feedback to facilitate students finding mistakes and building knowledge about how to proceed in case of errors. Moreover, the tool shows some potential to help students developing learning strategies, e.g. metacognitive awareness and improving problem-solving skills.

Keywords: Automatic assessment, formative evaluation, summative evaluation, computer programming, computer science education

INTRODUCTION

Learning to program computers is a process that requires a lot of practice work by students and a lot of feedback from the teacher [1][2]. The feedback activity must be based on the evaluation of the computer programs developed by students as this is a key activity to help students in their learning process [3]. The feedback activity must include summative and formative evaluation.

The summative evaluation indicates to the students if their solutions are satisfactory through a grade; while the formative evaluation gives them direct input about how they are doing the task, how they could do it better and how they could correct their programs [4]. Nevertheless, the evaluation of computer programs is not an easy task [5][6] since each program must be tested and its source code analysed considering syntax, semantics, efficiency and maintainability aspects [7]. The syntax can be verified automatically by a compiler/interpreter, but the evaluation of the other three aspects frequently is carried out manually, which makes the evaluation process tiresome and prone to faults [8].

For this problematic situation, several computer-based systems have been proposed to automatically evaluate programs and support the learning process of computer programming. Keuning et al classify these systems into two groups: automatic assessment tools (on-line judges or automatic grading tools) and learning environments for programming [9].

The first group, automatic assessment tools for computer programming assignments, is composed of software tools that automatically grade programming exercises by running the programs with different test cases [6][10-12]. In most cases, this kind of tool offers a summative evaluation to the students by means of quantitative grades, allowing them to identify some errors in their programs mainly related to syntax and functionality. However, most of these tools are not conceived as educational tools, but are used for competitive programming platforms. Although they provide summative feedback, they generally fall short as formative evaluation tools due to the limited feedback they provide to students.

The learning environments for programming include tools that support learning programming, instead of focusing on automatic evaluation processes [13-16]. Therefore, these tools offer formative feedback through different mechanisms that support the students during their learning process before they are evaluated, but generally these systems lack automatic grading tools. Consequently, there is a growing need for assessment tools that provide learners with formative feedback [9].

In this context, more research is needed to design automatic evaluation tools that not only support summative evaluation, but also provide some form of formative feedback during the evaluation. There is also a need to better understand the advantages and challenges of using these evaluation support tools in computer programming courses.

In this article, the authors describe the use of an automatic evaluation tool, called UNCode that supports both formative and summative evaluation in an introductory computer programming course [17]. Until now, the tool had not been used in an academic environment as a support system for programming courses. Therefore, the aim of this study is to report the experience of using UNCode in an academic context to explore its potential benefits for the student's learning process. This effort is motivated by two main research questions: first, how do students interact with UNCode? And second, what is the students' perception of UNCode as a support mechanism for their learning process?

EXPERIENCE REPORT

UNCode

The main goal of UNCode is to integrate an automatic grading system and several supporting functionalities to create a learning environment for computer programming. In other words, UNCode is an integral system that supports both kinds of feedback to students: formative and summative evaluation. It is a Web platform that integrates an automatic assessment tool, which facilitates performing a summative evaluation of programming assignments in a programming course, together with several modules from learning environments that contribute to the formative evaluation of the students. The following programming languages are supported: Java, Python and C/C++. A comprehensive description of UNCode was previously presented by the authors in a different work [17]. Unlike the previous work, which was focused on the tool, this article presents an experience report of the use of the tool in an academic context to explore its potential benefits for the learning process of the students. The tool is deployed at https://uncode.unal.edu.co/. Its source code is publicly available at https://github.com/JuezUN and it is distributed under an AGPL (GNU Affero General Public) 3.0 license.

The general UNCode workflow operates as follows. First, students access a particular programming assignment in the tool, where the description of the task is presented. Then, students write computer programs (source code), which are possible solutions for the programming assignment. During the development process, they can use different support functionalities that offer meaningful feedback (formative evaluation) based on their source code. Students can also submit their solution several times to the automated assessment tool, and receive the results indicating the possible presence of errors and in which test cases the program passed or failed (summative evaluation).

As an automated assessment tool for the summative evaluation, UNCode is built on top of INGInious [12], which is a secure and automatic grading system for programming exercises. In this way, summative (and immediate) evaluation is offered to students by means of the grades of their submissions (i.e. problem-solution attempts in the form of source code of a program). This feedback indicates whether the syntax, semantics and efficiency of the program are correct or have some type of inconvenience. The correctness of the solution is evaluated by running the program on different test cases. Based on the tool response, a grade is assigned to the program solutions proposed by the students.

Unlike most automated grading tools, UNCode also provides formative evaluation through different functionalities that support the learning process of the students when developing the solution of the programming exercises, even before the solutions are submitted for grading. These functionalities include syntax highlight, automatic verification of good programming practices (linter tips), code execution visualisation, custom input tests, comparison of expected program outputs against the obtained outputs and self-monitoring reports. Figure 1 presents several screenshots of the main functionalities of UNCode.

Syntax highlighting: this functionality consists of a rich code editor for UNCode that runs in the Web browser. It performs syntax highlighting for different programming languages, auto indentation and outdent and the possibility of handling large documents without trouble. In this way, students can develop their programs using the editor directly on the Web, which gives them the coding user experience of traditional integrated development environments (IDEs). Formative evaluation is provided to students as syntax errors are highlighted on the editor immediately, while they are writing the code. Thus, students do not have to wait until they submit their program solution to realise they have made a mistake related to the syntax of the programming language. A screenshot of the syntax highlighting functionality can be observed on the right of Figure 1a, where source code written in language C++ is highlighted.

Automatic verification of good programming practices (linter tips): UNCode has a linter, which is a software tool that analyses source code to flag programming errors, bugs, stylistic errors and suspicious constructs [18]. It is useful to recommend the best programming practices to students, while they are learning to code. This kind of tool helps students to deal with code maintainability issues in advance. It is worth mentioning that in many cases these issues are not addressed when automatic grading tools are used, even though code maintainability should be a required skill for every good programmer. In this sense, UNCode provides formative evaluation on code maintainability to students. An example of the recommendations given by the linter to a program developed by a student in C++ is presented on the left side of Figure 1a.

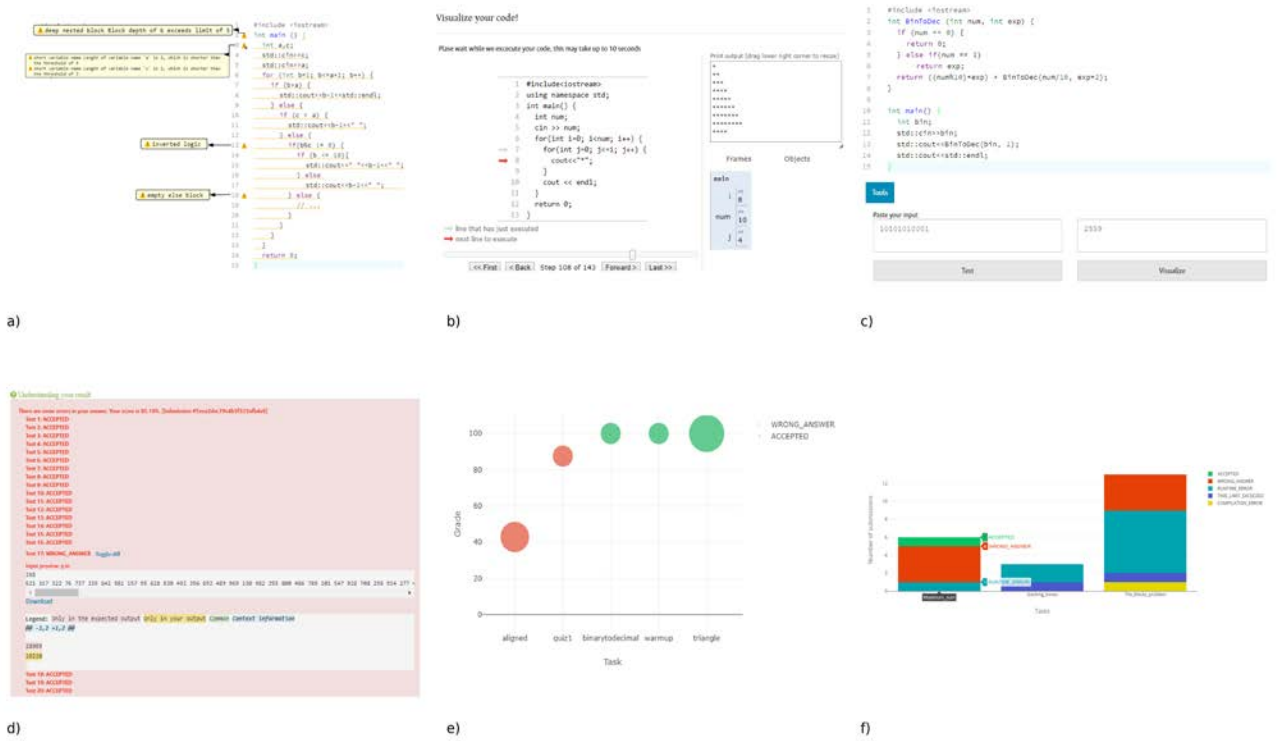

Figure 1: Screenshots of UNCode's formative evaluation functionalities: a) syntax highlighting and linter tips; b) code execution visualisation; c) custom input tests; d) comparison of expected outputs; e) self-monitoring reports: bubble chart of grades; and f) self-monitoring reports: stacked bar chart of submissions.

Code execution visualisation: UNCode also integrates a visualisation tool to see a graphical representation of the code execution: Python Tutor. Using this tool, students can write programs directly in the Web browser, and step forward and backward through the program execution, while visualising the run-time state of data structures and variables [19]. This may help students to learn computer programming allowing them to understand what happens as the computer executes each line of code in their programs. They can explore the visualisations interactively, at their own pace. Therefore, students receive graphical information about what they need to understand, clarifying how they are performing the programming tasks. Furthermore, not only Python code can be represented, but it also supports code in C/C++ and Java. For example, Figure 1b shows a screenshot of the visualisation of a simple program in C++ (step 108 out of 143).

Custom input tests: UNCode offers to students the possibility of testing their programs using their custom input test cases. These tests are not considered for grading, instead they become a tool for practising and testing as if they were working using a conventional IDE. The system executes the programs given the custom input test cases, and shows the obtained results to students. Hence, students can test the program inside UNCode before submitting it for grading, being able to trial a wide spectrum of input test cases ranging from basic input examples to complex corner cases. Figure 1c presents a screenshot showing the execution of a simple program (C++ code) that converts a binary number into its decimal representation. In this case, the binary number 10101010001 was typed in the textbox of the custom input, then by clicking the *test* button at the left bottom of the image, 2559 was obtained as a result (in the textbox at the right bottom).

Comparison of expected program outputs against the obtained outputs: unlike several automatic assessment tools, UNCode allows teachers deciding which test cases will be shown to students in case their program obtains a different output, in comparison with the expected output of the program, given a specific input test case. For each failing test case selected by the teacher, UNCode shows to the student the differences between the program output and the expected output, as well as the input of the test case. In many cases, this feedback can be valuable for the students, because they can identify problematic test cases for their solutions, and complete/correct their programs accordingly. Although this functionality is not desired in competitive programming environments, it can be meaningful in academic contexts. For example, Figure 1d presents a screenshot of the feedback obtained after grading a code submission. In this case, the partial grade obtained by the student was 85.19% (out of 100%). The feedback reports that the student's code got an incorrect answer in test case number 17. Therefore, the test case input is presented together with the differences between the expected and obtained results.

Self-monitoring reports: UNCode provides interactive statistical reports for both students and teachers. In this way, students can monitor their progress during the course. For example, they can observe a bubble chart showing the best grade obtained in each one of the programming assignments, where the size of the bubbles represents the number of submissions (attempts) that the student made to achieve that final grade (Figure 1e). Students can also visualise a stacked bar chart showing the distribution of the kind of feedback given to their submissions for every task (Figure 1f).

Target Course: Introduction to Computer Programming

Introduction to Computer Programming is a first-year course taught in the School of Engineering at the Universidad Nacional de Colombia for students of different engineering programmes. This course aims to develop algorithmic thinking skills in the students and the use of programming languages to solve computational problems in a systematic way. The course is equivalent to the CS1 (Computer Science 1) class in many other universities, and covers the basic concepts of imperative programming, including data types, variables, input/output, mathematical operations, conditional control structures, iterative control structures, functions, arrays and file management. The programming language used in the course is C++. The course lasts 16 weeks, which is the duration of the academic term.

The methodology of the course is based on two different types of weekly sessions: lecture sessions, where the professor presents the main concepts using different resources, such as slide presentations and live coding; and practical sessions, where the students solve individually different programming exercises in the computer laboratory. Moreover, the grades of the course are based on practical programming homework and examinations. The details about the class activities supported by UNCode are explained next.

Class Activities Supported by UNCode

The following activities were carried out with the support of UNCode to integrate the tool to the course methodology:

Introduction: the students were introduced to UNCode by the teaching staff. Each student was assigned an account and they tested the functionalities for accessing a problem, reading its statement, writing code, submitting a solution and receiving feedback from the tool. The students were also introduced to the supporting functionalities (formative evaluation) of UNCode that include real-time checking of coding best practices and potential errors, code execution visualisation, code testing with custom inputs, informative feedback on failed test cases and statistics of the student performance for self-monitoring their learning process. During this activity, the students had to solve a simple problem for training purposes, so that they tested all functionalities of UNCode, while solving the proposed problem guided by the teacher. This introductory activity was carried out during a single practical session of 120 minutes.

Homework: after the introductory demonstration, the students were given a programming task each week (i.e. homework) to solve with the support of UNCode. The deadline of each homework was due one week after its assignment. These activities were for home and were carried out individually by each student. In total, three programming assignments (homework) were given to the students during this study. The students had to submit their solutions through UNCode. The topics covered in the tasks corresponded to those being studied at the time in the course, including: variable declaration, input/output, mathematical operations, conditionals, loops, functions, arrays and matrices.

Examination: the students were given a single programming problem to solve during class time. They had to solve it individually and submit it using UNCode during a period of 60 minutes. They had to be physically present at the computer laboratory and could not do it remotely.

It is worth mentioning that the students could make several submission attempts for each programming problem. Activities (homework or examination) could be submitted any time before their deadline. The grade of the corresponding activity was obtained by considering the best attempt for each problem. In this experience, the number of incorrect submissions were not considered as part of the grade. This encouraged the students to try to solve the assignments by evolving their solutions incrementally, instead of finding a definitive solution in the first attempt.

PARTICIPANTS AND DATA COLLECTION

Participants

The participants in this experience were 56 first-year engineering students of the Introduction to Computer Programming course. They were selected based on a convenience sampling. The reason was that no probabilistic sampling was possible, since the study was performed in a course where the students had previously enrolled. The participants were informed about the goals of this research and they signed an informed consent before being included in this study. They were aware that their participation in the survey would not be considered for the grade of the course and that data collected in this study were confidential. On average, the participants were 19.5 years old at the time of completing the survey, with a standard deviation of 3.6 years. The majority of them were male (85.7% of the participants). The students belonged to different engineering programmes, including: mechanical engineering (35.7%), electronic engineering (21.4%), electrical engineering (12.5%), civil engineering (12.5%), computing engineering (5.4%), and others (12.5%).

Data from the Students' Interaction with UNCode

To answer the first research question - how do students interact with UNCode? - relevant data was collected over five weeks, during which the students developed classroom and home activities supported by UNCode. The data collection was carried out using UNCode's automated tools and data analysis was performed using descriptive statistics.

During this experience, the participants performed five different activities using UNCode as a support tool: introduction to UNCode, three homework activities and one examination. Data were collected about the students' interaction with the tool during the development of each activity, including student participation in each activity and data related to the performance of the students in the corresponding programming assignments. These data were analysed through descriptive statistics, and the results are presented in the article. Regarding student participation, the number of students who took part in each activity was registered. The number of attempts made, when trying to solve the programming assignment, was also considered.

In each activity, data were collected for the performance of the students in each programming assignment. Every attempt of solution was graded automatically by UNCode, informing whether the proposed solution was successful or not. The successful solutions were categorised as *accepted*. In the case of a failed attempt, UNCode identified the type of error, and it was classified as one of the following: *compilation error*, *runtime error* or *wrong answer*. The compilation error occurs when the source code of the solution is incorrect at lexical or syntactic levels. The runtime error indicates that there was an exception during the program run-time and the execution was not properly finished; for example, division by zero, access to an invalid memory location, etc. Finally, the *wrong answer* error shows that the program is incorrect at the semantic level, i.e. it produced an output different than the one expected in a particular test case.

Data from the Students' Perception Regarding UNCode

In addition, at the end of the experience, to answer the second research question - what is the students' perception regarding UNCode as a support mechanism for their learning process? - the authors used a survey with both multiple-choice questions and open-ended questions. The students filled out the survey on their perceptions of UNCode and its functionalities. They completed the survey after using the tool for five weeks. The survey was used to analyse different aspects of the students' perceptions, including the usefulness of UNCode and each of its functionalities for the learning process, and the utility of the automatic system for grading programming tasks.

To collect the data, an on-line form was prepared with the survey using Google Forms and all the students were asked to complete it. Participation was voluntary, and they were informed that the goal of the survey was to get to know their perceptions about UNCode, and the data were confidential and treated anonymously. It is important to mention that the students were told that their answers, whether positive or negative, would not affect their grades of the subject. In terms of the students' perception of UNCode, the following multiple-choice statements were included in the survey: 1) UNCode was useful for my learning process; and 2) UNCode is a useful tool to evaluate the Computer Programs I developed in this class.

The possible options for the statements consisted of a Likert scale with six levels of agreement/disagreement: totally disagree 1, disagree 2, somehow disagree 3, somehow agree 4, agree 5 or totally agree 6. Moreover, for each one of the statements, an open-ended question was asked to the students to get to know why they select that level of agreement/disagreement. These questions were not optional to fill out the survey. In this way, not only was it possible to get to know the different levels of agreement of the students, but also to know their detailed perceptions about UNCode.

In addition, the students were asked to rate each one of the following UNCode functionalities according to the utility they have, in their opinion, to learn computer programming: automatic verification of good programming practices (linter), custom input tests, syntax highlight, code execution visualisation (Python Tutor), comparison of expected program outputs against the obtained outputs, self-monitoring reports. The level of agreement/disagreement of the students was measured using the same answer scale as the one used for the previous statements, except that in this case, there was an additional option: *NA - not applicable*. The NA option was conceived to be selected when the students were not familiar with the descriptive name of the UNCode's functionality.

Results for the multiple-choice questions of the survey were analysed through descriptive statistics. Bar charts were made in order to analyse the number of students that selected each level of agreement of the Likert scale for each statement. On the other hand, the procedure for analysing the answers to the open-ended questions was conducted according to the recommendations for data qualitative analysis [20]. Initially, the answers given by the students were encoded. The codification of the responses made it possible to obtain indicators showing the impact of UNCode on their learning process. These indicators included: learning activities, learning achievements that students considered they attained with the use of UNCode and mental processes carried out by students that were activated by the tasks performed, among others. Codes and indicators were then analysed to establish common aspects and links between them. Codes and indicators were grouped, and the resulting groups were associated with concepts.

Finally, the concepts were analysed to determine common characteristics among them and to establish groups that could be used to obtain categories that would globally represent students' perceptions of their learning experience with UNCode. The process was carried out in an iterative manner, which means that the naming of codes, the establishment of indicators and the definition of concepts was done several times and making the necessary adjustments to give the greatest possible coherence to the interpretation of the students' perceptions. Additionally, it is worth mentioning that the analysis of the qualitative data was carried out by two of the main researchers in this work with the help of digitised tables in which the perceptions given by the students were transcribed and ordered, and the whole process described above was carried out.

RESULTS AND DISCUSSION

Student-Tool Interaction Analysis

Table 1 presents the number of students who participated in each different class activities using UNCode as a support tool during this experience. It also includes the total number of submissions carried out by them. A student is considered to participate in an assignment if he/she performed at least a single submission during the activity. The students' participation might vary because not always they got to make a submission, even though they were trying to solve the programming assignments. In these cases, the students were not considered as participants.

Table 1: Number of participating students and submissions by programming assignment.

Activity	Students	Submissions
Introduction	39	316
Homework 1	31	244
Homework 2	49	282
Homework 3	37	277
Examination	49	136

On average, the number of participants per task was 41 (out of 56 - 73.2%). Moreover, the students performed 251 submissions per task on average. And more importantly, the average number of submissions per participant was 7.3 for the programming assignments from the following class activities: introduction and homework; while this average was 2.8 for the examination. This was mainly due to the duration of the examination being only 60 minutes long, whereas the introductory activity took 120 minutes, and the homework activities were carried out throughout the week.

Figure 2 illustrates the students' performance during the activities carried out. Each bar represents the total number of submissions per class activity. The number of submissions in each category is shown according to their feedback obtained: accepted, compilation error, runtime error and wrong answer. Note that only the first category corresponds to a desirable result.

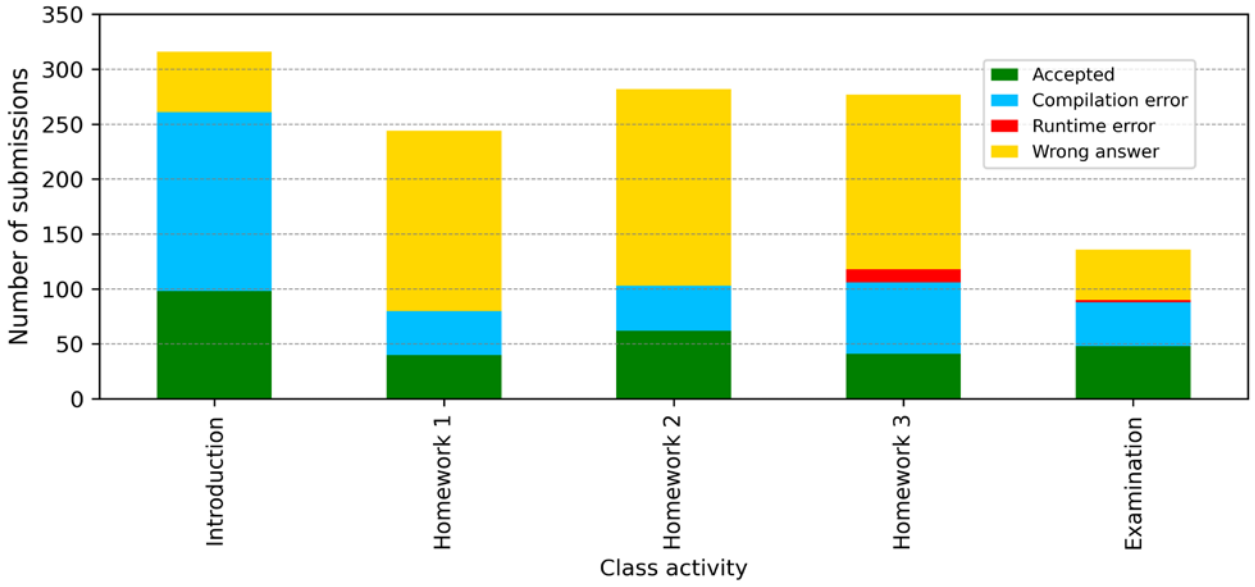

Figure 2: Number of submissions in each category (accepted, compilation error, runtime error and wrong answer).

In the case of the introductory activity, there were 98 submissions considered as correct, 163 catalogued as compilation error and 55 attempts classified as wrong answers, i.e. 316 submissions in total. It is worth noting that each student is free to make several submissions. That is why the number of correct submissions is greater than the number of participants in the activities. In this case, although 98 submissions were considered as correct, the number of participants was only 39 students, of which 35 of them performed at least one correct submission. In addition, one can observe another particularity regarding the high number of errors (163 compilation errors and 55 wrong answers). This can be explained by the novelty of the tool to the students, and at the same time, the purpose of the activity, which was to train the students in the use of UNCode as a support tool for the assessment of their tasks. Thus, it was necessary to show them different kinds of errors and their corresponding feedback.

Regarding the homework activities, the large number of submissions shows that the students used the tool considerably. This is important considering that these assignments were developed as home activities, which denotes the engagement of the students to achieve the goal of the proposed tasks in an independent way. Furthermore, the number of submissions classified as correct, compilation error and wrong answer, were consistent along the three homework activities. On average, 17.7% of the submissions were catalogued as correct (this percentage corresponds to an average of 29 unique students who obtained at least one correct submission, i.e. without considering multiple correct submissions), 18.1% was considered as a compilation error and 62.7% of the submissions were classified as wrong answers. These results show that the difficulty of the proposed homework was similar, and at the same time, that the students' dedication was comparable among the different homework.

The topics covered in each homework were as follows: homework 1 included basic aspects of the programming language and conditionals; homework 2 contained numeric calculations and loops; and homework 3 evaluated the topic of functions. Although each homework was devoted to a specific topic, the proposed tasks required an accumulative understanding of the previous topics. For instance, in homework 3, students should demonstrate proficiency not only in functions, but also in loops, conditionals and the basic use of the language. That is, homework 3 required more concepts to consider, and their attempt of solutions could result in a few special errors, such as runtime exceptions (e.g. segmentation faults, floating-point error, etc). In this case, the percentage of runtime error was 4.3%, i.e. 12 submissions.

Finally, with respect to the examination, not only was it possible to notice that the number of submissions was lower due to the time restriction of the activity itself, but also the proportion of wrong answers was reduced down to 33.8%. This percentage is close to half of the wrong answer percentage of the homework activities (62.7% of the submissions). In addition, the percentage of submissions catalogued as correct increased up to 35.3%, which is almost twice the usual proportion of correct submissions in the homework activities (17.7% of the submissions). However, considering absolute values, the total number of correct submissions in the examination was 48, and the average number of submissions in the homework activities was 47.6. The 48 correct submissions of the examination were obtained by 28 unique students. This means that the students achieved a similar number of correct submissions, but they did not need a large number of attempts in the examination in comparison to the number of attempts required in the homework activities. These results can be explained considering that some students have trained with the homework activities during the previous weeks before taking the examination, achieving an expertise level in the topics covered in the examination. Additionally, this might show that the brute-force approach to solving the examination questions is not a good strategy, because of the time limitation, which fosters students to analyse their solutions thoroughly before they perform a submission.

Students' Perception Analysis

The survey was used to analyse different aspects of the student perceptions, including: usefulness of UNCode for the learning process, utility of the automatic system for grading programming assignments and the usefulness of each one of the functionalities of UNCode to learn computer programming.

Usefulness of UNCode for the Learning Process

Regarding the usefulness of UNCode for the students' learning process, Figure 3 (top bar of the chart) shows the levels of agreement/disagreement expressed by the respondents in a Likert scale using a horizontal stacked bar chart. The results of the survey showed that 76.8% of the respondents (43 out of 56 students) considered that UNCode was useful for their learning process with different levels of agreement. Although the remaining percentage of students (23.2%) indicated some level of disagreement, none of them with the highest level of disagreement.

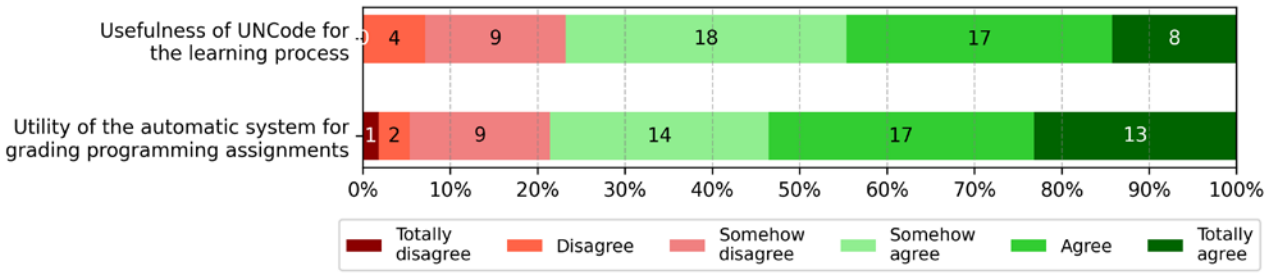

Figure 3: Levels of agreement/disagreement expressed by the students with respect to the usefulness of UNCode for their learning process and the utility of the automatic system for grading programming assignments.

A qualitative analysis was also performed from the 56 answers to the open-ended question that was directed to the students to get to know why they selected that level of agreement/disagreement. Five categories were identified to classify the answers of the students who consider that UNCode was useful for their learning process. These categories are:

Functionalities to debug programs: some students opined that several functionalities of UNCode were convenient to verify the correctness of the programs not only from the syntactic point of view, but also from the maintainability and

functionality aspects. They indicated that the UNCode's functionalities of syntax highlighting and automatic verification of good programming practices (linter tips) help them to detect errors or possible improvements in the maintainability of the source code, while they were writing it. In addition, some students stated that the possibility to compare the expected program outputs against the obtained outputs of their programs gave them the opportunity to detect malfunctioning code and correct semantic errors at a functional level. In total, 30 answers of the students were classified within this category. For instance, one of them expressed: *Not only it is restricted to being a code writing tool, but it also helps the user to know what mistakes they are making, in addition to giving advice for bad habits (in programming)*, and another one wrote: *Since it provides different applications to know where the program failed or if everything is fine*. Note that the 30 answers classified in this category are not exclusive, and they can be classified within other categories as well.

Frequent practice of programming: some students also expressed positive opinions regarding the possibility to carry out frequent practices on programming assignments. In particular, 17 total answers were identified in that regard. Among them, two main topics were often expressed: they could practise the concepts from the lectures, and they were able to deepen their learning of programming through the practice. The first one included opinions with respect to the benefits of the course methodology and the pertinence of the problems proposed in the homework activities. For example, a student wrote: *The different exercises proposed on the platform were appropriate for the level of difficulty and to practise what was read and seen in class*. The second main topic encloses perceptions on the advantages given by the support from UNCode to the learning process. For instance, one of the students expressed: *Thanks to UNCode I worked much more and emphasised issues that I was not so clear about*.

Problem solving: some students (in seven answers) referred to the usefulness of UNCode to help them to improve their problem-solving skills. They noted that thanks to the tool they could achieve a deeper understanding of the problems in the homework, and explore ideas and possible solutions to design and implement a program to solve the problems. Two of them stated: *...it allows the problems to be very well understood, thus facilitating the process of solving them...*, *…because it encourages the use of acquired knowledge to solve problems in a creative and varied way*.

Availability of the on-line tool: some opinions (six answers) of the students highlighted the importance of the on-line continuous availability of UNCode to carry out the programming assignments. This gives them the opportunity to practise and learn during home activities from any location without depending on special software requirements. For example: *As it is an on-line tool, it helps the student to practise without the need to have programs installed; one can learn anywhere*.

Metacognition: as a cross-cutting issue, the students also expressed some opinions related to metacognition in programming, i.e. they were aware of their thinking and learning process of programming. Metacognition usually refers to planning, monitoring and assessing one's understanding and performance in the learning process [21], and in this case, the students (seven answers) opined that the use of UNCode allowed them to monitor and assess their learning of programming. For example, one of them said: *(UNCode) is a support system to assess and know the level of knowledge I have about the subject*, and another one indicated: *...(UNCode) has many tools in which a user can improve his way of programming, and also understand what happens when code is executed*.

Utility of the Automatic System for Grading Programming Assignments

With respect to the utility of the automatic grading systems for programming assignments, the answers of the students are illustrated in Figure 3 (bottom bar of the chart). It shows the levels of agreement/disagreement expressed by the respondents on a Likert scale. The results of the survey showed that 78.6% of the respondents (44 out of 56 students) considered that the automatic system for grading programming assignments was useful with different levels of agreement. Only 21.4% of the students indicated some level of disagreement, including only three students with the two highest levels of disagreement.

From the qualitative analysis of the answers to the open-ended question that was directed to the students to inquire why they chose that level of agreement/disagreement, it was possible to identify two main categories to group the answers:

Feedback and testing for programming assignments: 26 different answers of the students with their perceptions allow the researchers to identify that the most valued features of the automated grading tool are: immediate feedback, formative feedback and testing capabilities. Firstly, it provides immediate feedback, which is useful for the students, because it permits them to obtain timely feedback, simultaneously while they are solving the programming assignments. For instance: one of the students affirmed: *…in general, it facilitated the evaluation making it faster and more objective*.

Secondly, the students considered that the tool provided formative feedback, giving them directions about how they are doing the task, what elements they should revise to do it better and facilitating to find errors in the programs. For example, one of the students said: *…it is a simple study and evaluation tool that allows us to find errors in the programs to be developed*. Finally, some students highlighted the testing functionalities to verify the correctness of their programs. Many of them considered that the comparison of expected program outputs against the obtained outputs was a useful functionality, which facilitated to verify their programs and find possible errors based on varied test cases. An example of this perception is expressed as follows: *…it helps me to know the result of the program with different test cases*.

Improvements needed for UNCode: among students' opinions it was possible to group nine answers into a category of recommendations of potential improvements for future versions of the tool. It is worth noting that these recommendations were expressed by some students that considered that UNCode was useful for the evaluation of programming assignments but needed some improvements. The students' recommendations included: to provide more details and explanations regarding errors in the programs evaluated by UNCode; and to offer automatic alternative solutions to errors. The following two sentences are examples of the students' answers in this regard: …*it is a good tool; however, it is not clear enough regarding the errors made in the programs. Although it provides an automatic grade, it does not help in finding alternative solutions to programming errors.*

Usefulness of Each One of the Functionalities of UNCode to Learn Computer Programming

Figure 4 groups the rates given by the students to each of the UNCode's functionalities to support the learning of computer programming. It presents the levels of agreement/disagreement expressed by the respondents in a Likert scale using a horizontal stacked bar chart per each functionality: syntax highlight, automatic verification of good programming practices (linter tips), code execution visualisation, custom input tests, comparison of expected program outputs against the obtained outputs and self-monitoring reports. Besides the usual options in the Likert scale, there was an additional option: *NA - not applicable* that was conceived to be selected when the students were not familiar with the descriptive name of the UNCode's functionality.

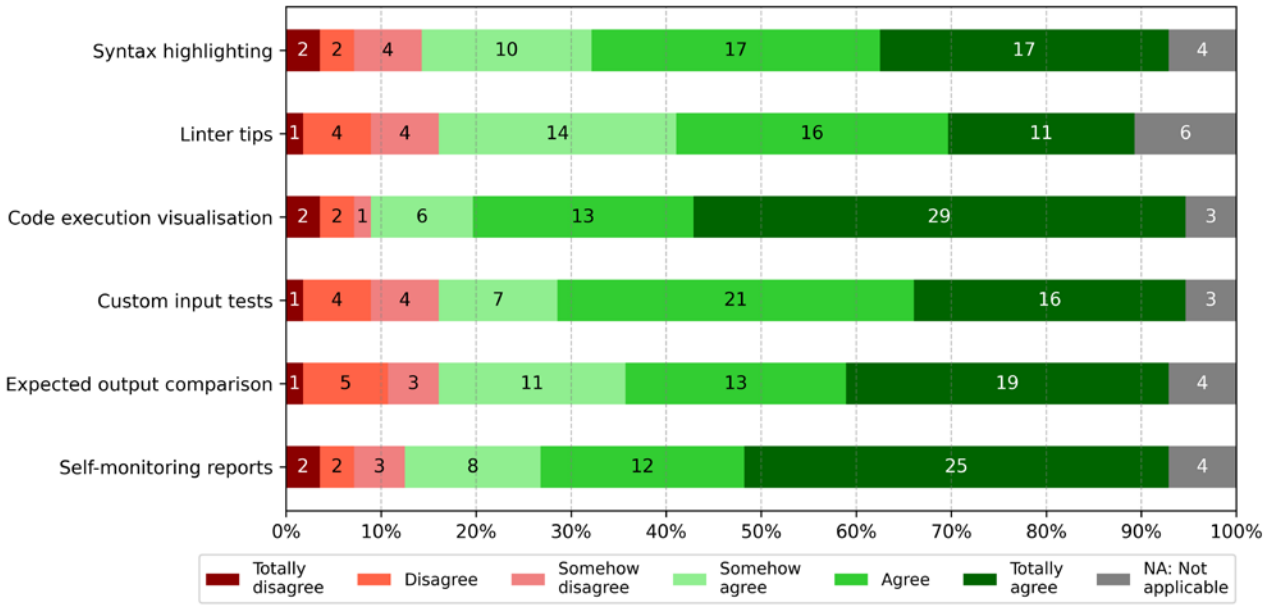

Figure 4: Levels of agreement/disagreement expressed by the students regarding the usefulness of each one of the functionalities of UNCode to learn computer programming.

Among the 56 respondents, most students rated several UNCode's functionalities to learn computer programming in a positive way, with the majority of their rates classified in one of the levels of agreement (4 - somehow agree; 5 - agree; or 6 - totally agree). The percentage of students that rated the functionalities in these levels of agreement was between 73.2% (linter tips) and 85.7% (code execution visualisation), for all functionalities. In contrast, some students expressed some level of disagreement with the usefulness of the UNCode's functionalities to support their learning process. The percentage of them ranged from 8.9% (five students) to 16.1% (nine students). The rest of the students selected the *NA: not applicable* option, ranging between 5.4% (tree students) and 10.7% (six students).

From the qualitative analysis carried out in regard to the students' answers to the open-ended questions of the survey, it was possible to find several mentions to concrete UNCode's functionalities in an indirect way. Most mentions were mainly related to the following functionalities: automatic verification of good programming practices (linter tips), custom input tests and comparison of expected program outputs against the obtained outputs. For example, in the first case, with respect to the usefulness of the linter tips, one of the students said: …*it was useful for learning because it gave quite useful advice while writing code*; regarding the custom input tests, another one stated: …*I was able to test the codes that I was writing, and receive immediate feedback*; and with respect to the possibility to compare expected outputs against the obtained outputs, one student opined: ...*functionalities, such as the analysis of expected output with respect to the output of my program and the Linter allow a clearer analysis of possible errors in the program.*

Discussion

This experience report was motivated by two main questions: first, how do students interact with UNCode? And the second, what is the students' perception regarding UNCode as a support mechanism for their learning process? The reported results from previous sections helped to answer these questions.

Regarding the first question, how do students interact with UNCode as a support tool in a computer programming course, the findings can be summarised as follows: the possibility that the students can perform multiple attempts to solve the programming assignments obtaining immediate feedback from the tool was advantageous, because it allows them to practise computer programming tasks frequently. The number of submissions was one order of magnitude higher than the number of participants in the study. The methodology of the course supported by UNCode promotes the submission of multiple attempts to get feedback from the tool, thus students receive feedback on the task each time they make a submission. In case of obtaining feedback informing an error in the proposed solution, students can identify aspects that need corrections, with multiple opportunities to submit new versions of their solution, which is an important advantage.

Contrarily, without the automatic assessment tool, students would have a limited number of opportunities to submit their solutions for evaluation and the teaching staff would not be able to cope with this amount of submissions to be graded manually. In this way, by using UNCode, it is possible to provide timely and objective feedback for students, and it also facilitates them to practise programming tasks frequently and continually. These findings support at least three of the advantages of automated assessment tools pointed out by Gordillo [22]. First, it is beneficial for students, because it increases their motivation, which is evidenced in the large number of submissions. Second, it improves the quality of their programs, which can be reflected in the fact that students can identify aspects that need corrections every time they obtain feedback from the tool. Third, the tool enhances their practical programming skills as they obtain timely and objective feedback while practising regularly.

Regarding the students' perception of UNCode as a support mechanism for their learning process, the findings indicate that the benefits of the tool for the formative and summative evaluation in a computer programming course include: functionalities to debug programs, possibility to frequently practise programming, features for the automatic feedback and testing for programming assignments, and the availability of the on-line tool. In addition, the most valued functionalities of UNCode for the formative evaluation during the learning process include features that are not common in traditional automatic grading tools like: code execution visualisation, automatic verification of good programming practices (linter tips), custom input tests and comparison of expected program outputs against the obtained outputs. In this sense, UNCode is not only useful as an automatic grading tool that provides summative evaluation for computer programming assignments, but it is also a tool that offers formative evaluation through different mechanisms that support students during their learning process, thus facilitating them to find and correct errors. In this sense, this experience report contributes to the field by offering evidence of a tool (UNCode) that supports students during their learning process and provides them with functionalities and information to build knowledge about how to proceed in case of errors [23].

Moreover, some additional potentialities of the tool were identified according to the students' perceptions: UNCode helps students to develop problem-solving skills in computer programming, and also helps them to be aware of their thinking and learning process of programming, i.e. metacognition as a learning strategy. On the one hand, students can achieve a deeper understanding of the proposed problems and their possible solutions, which is a major learning difficulty related to problem-solving skills in computer programming [24]. On the other hand, metacognition is important as the transformation from novice to expert programmers who are guided by self-reflection on previous success, or in many cases failure, by identifying specific issues for improvement, and then they can follow by frequent and conscious practice to address these issues [25][26].

In this way, tools like UNCode can provide implicit support for students to develop learning strategies related to metacognitive awareness and problem-solving skills, by means of the learning environment and the information that the tool provides [27]. This experience report contributes to the understanding of the potential benefits that a support tool for the formative and summative evaluation in a computer programming course can offer in an academic context, thus alleviating several typical difficulties from the learning process of computer programming.

CONCLUSIONS AND FUTURE WORK

This article described the use of an automatic tool (UNCode) that supports both formative and summative evaluation, in an introductory computer programming course. UNCode's functionalities include not only automatic assessment features for computer programs, but also several functionalities to support the students' learning process, while they are attempting to develop the solution to a programming problem. The results of this study helped to discuss two research questions: how did students interact with UNCode? And what was the students' perception regarding UNCode as a support mechanism for their learning process? The first question was answered using quantitative analysis on the interaction of the students with the tool, and the second question through qualitative analysis of an open-ended questionnaire, where the students expressed their opinions and perception about UNCode and its potential.

The students perceived UNCode as a valuable resource for their learning process, particularly due to the possibilities it offers to debug programs, to carry out programming as a frequent practice, features for the automatic feedback and testing of programming assignments, and its availability as an on-line tool. This is also shown by the wide students' interaction with UNCode that evidenced advantages, like obtaining automatic and immediate feedback from the tool and practising computer programming tasks frequently. Students also opined that they received implicit support to develop learning strategies, such as metacognitive awareness, and to improve problem-solving skills.

The mentioned findings endorse some of the advantages of automated assessment tools pointed out in other studies and contribute to the field by offering evidence of the particular benefits of the tools that support students during their learning process. In particular, the findings of this study suggest that for automated assessment tools it is advisable to integrate summative and formative evaluation to help students not only to identify errors, but also to build knowledge about how to proceed in case of errors.

Regarding the threats to the validity of this experience report, one limitation is the duration of the study, which reports an experience of five weeks of the course. A longer study requires a full intervention of the course adapting contents and methodologies to use the tool. A short study helped to justify a full intervention, which will be the goal of a future study. Another limitation is the external validity of the study. Although the participants in this study were enrolled in different engineering programmes, it is important to note that they belong to a single school of engineering; in other words, this is not a cross-institutional study. The findings offer valuable evidence of the advantages of automated assessment tools in particular learning scenarios, but the study design used in this work does not allow extending the conclusions to more general scenarios.

For future work, it is important to explore wider research questions about UNCode in cross-institutional studies, with students from different knowledge areas not only from engineering, using other study designs that include control and experimental groups for a longer period of time. In addition, from the technical point of view, future work will include other alternatives in the feedback provided to students according to their suggestions during this experience.

REFERENCES

1. Alhazbi, S., Using e-journaling to improve self-regulated learning in introductory computer programming course. *Proc. IEEE Global Engng. Educ. Conf.*, 352-356 (2014).
2. Robins, A., Rountree, J. and Rountree, N., Learning and teaching programming: a review and discussion. *Computer Science Educ*., 13, **2**, 137-172 (2003).
3. Restrepo-Calle, F., Ramírez Echeverry, J.J. and González, F.A., Continuous assessment in a computer programming course supported by a software tool. *Computer Applications in Engng. Educ*., 27, **1**, 80-89 (2019).
4. Souza, D.M., Felizardo, K.R. and Barbosa, E.F., A systematic literature review of assessment tools for programming assignments. *Proc. IEEE 29th Inter. Conf. on Software Engng. Educ. and Training*, 147-156 (2016).
5. Fonte, D., Da Cruz, D. Gançarski, L. and Henriques, P.R., A flexible dynamic system for automatic grading of programming exercises. *Proc. 2nd Symp. on Languages, Applications and Technologies*, 29, 129-144 (2013).
6. Cheang, B., Kurnia, A., Lim, A. and Oon, W.C., On automated grading of programming assignments in an academic institution. *Computers & Educ.*, 41, **2**, 121-131 (2003).
7. Carter, J., Ala-Mutka, K., Fuller, U., Dick, M., English, J., Fone, W. and Sheard, J., How shall we assess this? *Proc. ITiCSE-WGR '03 Working Group Reports from ITiCSE on Innov. and Technol. in Computer Science Educ*., 35, **4**, 107-123 (2003).
8. Ala-Mutka, K.M., A survey of automated assessment approaches for programming assignments. *Computer Science Educ*., 15, **2**, 83-102 (2005).
9. Keuning, H., Jeuring, J. and Heeren, B., Towards a aystematic review of automated feedback generation for programming exercises - extended version. *Proc. 2016 ACM Conf. on Innov. and Technol. in Computer Science Educ*., 41-46 (2016).
10. Ihantola, P., Ahoniemi, T., Karavirta, V. and Seppälä, O., Review of recent systems for automatic assessment of programming assignments. *Proc. 10th Koli Calling Inter. Conf. on Computing Educ. Research - Koli Calling '10*, 86-93 (2010).
11. Caiza, J.C. and Del Álamo Ramiro, J.M., Programming assignments automatic grading: review of tools and implementations. *Proc. 7th Inter. Technol., Educ. and Develop. Conf*., 5691-5700 (2013).
12. Derval, D., Gego, A., Reinbold, P., Frantzen, B., Van Roy, P. and De Louvain, U., Automatic grading of programming exercises in a MOOC using the INGInious platform. *Proc. European MOOC Stakeholder Summit 2015*, 1, **12**, 86-91 (2015).
13. Guzdial, M., Programming environments for novices. *Computer Science Educ. Research*, 127-154 (2003).
14. Gómez-Albarrán, M., The teaching and learning of programming: a survey of supporting software tools. *The Computer J*., 48, **2**, 130-144, (2005).
15. Le, N.-T., Strickroth, S., Gross, S. and Pinkwart, N., *A Review of AI-Supported Tutoring Approaches for Learning Programming*. Heidelberg: Springer, 267-279 (2013).
16. Nesbit, J.C., Liu, L., Liu, Q. and Adesope, O.O., Work in progress: intelligent tutoring systems in computer science and software engineering education. *Proc. 2015 ASEE Annual Conf. & Expo*., 26.1754.1-26.1754.12 (2015).
17. Restrepo-Calle, F., Ramírez-Echeverry, J.J. and Gonzalez, F.A., UNCode: interactive system for learning and automatic evaluation of computer programming skills. *EDULEARN18 Proc*., 1, 6888-6898 (2018).
18. Darwin, I.F., *Checking C Programs with Lint*. O'Reilly & Associates (1990).
19. Guo, P.J., Online python tutor: embeddable web-based program visualization for cs education. *SIGCSE '13: Proc. 44th ACM Technical Symp. on Computer Science Educ*., 579-584 (2013).
20. Bryman, A., *Qualitative Data Analysis in Social Research Methods*. (5th Edn), Oxford: Oxford University Press (2016).

21. Pintrich, P., Smith, D., García, T. and McKeachie, W., *Pintrich (1991) Manual for use of the Motivated Strategies for Learning Questionnaire*. 1-75 (2002).
22. Gordillo, A., Effect of an instructor-centered tool for automatic assessment of programming assignments on students' perceptions and performance. *Sustainability*, 11, **20**, 5568 (2019).
23. Ullah, Z., Lajis, A., Jamjoom, M., Altalhi, A., Al-Ghamdi, A. and Saleem, F., The effect of automatic assessment on novice programming: strengths and limitations of existing systems. *Computer Applications in Engng. Educ*., 26, **6**, 2328-2341 (2018).
24. Costa, C.J. and Aparicio, M., Evaluating success of a programming learning tool. *ISDOC 2014 Proc. Inter. Conf. on Infor. Systems and Design of Communic.*, 73-78 (2014).
25. Loksa, D., Ko, A.J., Jernigan, W., Oleson, A., Mendez, C.J. and Burnett, M.M., Programming, problem solving, and self-awareness: effects of explicit guidance. *Proc. 2016 CHI Conf. on Human Factors in Computing Systems*, 1449-1461 (2016).
26. Caruso, T., Hill, N., Van DeGrift, T. and Simon, B., Experience report: getting novice programmers to THINK about improving their software development process. *SIGCSE'11: Proc. 42nd ACM Technical Symp. on Computer Science Educ.*, 493-498 (2011).
27. Prather, J., Pettit, R., McMurry, K., Peters, A., Homer, J. and Cohen, M., Metacognitive difficulties faced by novice programmers in automated assessment tools. *ICER '18: Proc. 2018 ACM Conf. on Inter. Computing Educ. Research*, 41-50 (2018).

BIOGRAPHIES

Felipe Restrepo-Calle received his Bachelor's degree in systems and computer engineering and the MSc degree (*cum laude*) in physics instrumentation from the Universidad Tecnológica de Pereira, Colombia, in 2004 and 2007, respectively, and the PhD degree (*cum laude*) from the University of Alicante, Spain, in 2011. He was a post-doctoral research fellow at the University of Seville, Spain, from 2012 to 2013, and the Universidade Federal do Rio Grande do Sul, Porto Alegre, Brazil, during the first semester of 2014. He is currently an assistant professor in the Department of Systems and Industrial Engineering at the Universidad Nacional de Colombia, Bogotá. His fields of research interest include programming languages, dependable design in embedded systems and engineering education.

Jhon Jairo Ramírez-Echeverry received his Bachelor's degree in electronics engineering from the Universidad Nacional de Colombia, Manizales (Caldas), the MSc degree in telecommunications engineering from the Universidad Nacional de Colombia, Bogotá, and the PhD degree (*cum laude*) in engineering of projects and systems from the Universitat Politècnica de Catalunya - BarcelonaTech, in 2017. He is currently an associate professor in the Department of Electrical and Electronics Engineering at the Universidad Nacional de Colombia, Bogotá. His research interest areas are engineering education (self-regulated learning) and electronics telecommunications systems.

Fabio Augusto González Osorio received his Bachelor's degree in computing systems engineering, the MSc degree in mathematics from the National University of Colombia, Bogotá, and the MSc and PhD degrees in computer science from the University of Memphis. He is currently a full professor in the Department of Computing Systems and Industrial Engineering at the National University of Colombia, where he leads the Machine Learning, Perception and Discovery Laboratory. His research interests revolve around machine learning and its applications in information retrieval, computer vision, natural language understanding, and biomedical image analysis, with a particular focus on the representation, indexing and automatic analysis of multimodal data (data encompassing different types of information - textual, visual, signals).