Types of assessing student-programming knowledge

Anabela Gomes
Engineering Institute of Polytechnic Institute of Coimbra
Centre for Informatics and Systems – University of Coimbra
Coimbra, Portugal
anabela@isec.pt

Fernanda Brito Correia
Engineering Institute of Polytechnic Institute of Coimbra
Institute of Electronics and Informatics Engineering of Aveiro (IEETA) - University of Aveiro
Coimbra and Aveiro, Portugal
fernanda@isec.pt

Pedro Henriques Abreu
Faculty of Sciences and Technology
Centre for Informatics and Systems – University of Coimbra (CISUC)
Coimbra, Portugal
pha@dei.uc.pt

***Abstract*— High failure and dropout rates are common in higher education institutions with introductory programming courses. Some researchers advocate that sometimes teachers don't use correct methods of assessment and that many students pass in programming without knowing how to program. In this paper authors describe the assessment methodology applied to a first year, first semester, Biomedical Engineering programming course (2015/2016). Students' programming skills were tested by playing a game in the first class, then they were assessed with three tests and a final exam, each with topics the authors considered fundamental for the students to master. A correlation analyses between the different types of tests and exam questions is done, to evaluate the most suitable, for assessing programming knowledge, showing that it is possible to use different question types as a pedagogical strategy, to assess student difficulty levels and programming skills, that help students acquire abstract, reasoning and algorithm thinking in an acceptable level. Also, it is shown that different forms of questions are equivalent to assess equal knowledge and that it is possible to predict the ability of a student to program at an early stage.**

Keywords— Introductory programming; Programming learning difficulties, Programming assessment, CS1

I. INTRODUCTION

Introductory programming courses at our Institution with stand the same problems reported in many other higher education institutions: high failure and dropout rates are common in those courses [1-4]. This is a situation that affects essentially freshmen as those courses are usually placed at the beginning of the curricula [1, 5, 6]. Many reasons were already identified [4, 7-10]. Those difficulties are mainly related with the students' background knowledge, the nature of programming, the learning and study methods and the pedagogical strategies commonly used in introductory programming courses. In this context different types of tools have been developed, including special purpose integrated development environments (IDEs), microworlds for learning, algorithm animation systems, among other tools [11-13]. In [14] there is a study describing preliminary results of research related to programming teaching tools. Some visualization tools, which were intended to help students in their early programming stages, have already been developed by some elements of this team [15].

There are also several teaching and learning approaches using different methodologies such as flipping classroom [16], coding tools and different techniques [17], student self-direction technologies [18], extreme apprenticeship [19] or others [20-22]. Recent technologies in the context of the classroom such as the ones using tablet PCs, Personal Digital Assistants or Arduino are also worth mentioning [23-26]. It is also particularly interesting to mention the use of robots, like LEGO Mindstorms robots and other types of robots to support programming learning with different purposes and in diverse contexts [27-31]. Different types of game approaches, specially the ones based in serious games have also been developed to engage students in the process of learning programming in a more enjoyable and motivational way [32-37].

However, the results of their use didn't change the panorama substantially. For some time we have been studying the problems associated with the teaching and learning of the first programming subject and also proposed solutions [38]. From 2011/2012 onwards we have been attempting several other approaches in order to minimize these problems. We started by registering and analysing the type of errors students make when programming (during classes and exams) [39]. In order to minimize them, we made a set of recommendations for teachers and students, about some strategies that have shown results in order to surpass or avoid some problems. There are some students with very good marks; nevertheless the final success rate is still low. Therefore we did a study [40] on the patterns used by the students with higher and lower marks. The idea was the dissemination of good programming learning behaviours and the avoidance of bad ones.

Another subsequent change done was related with the type of assessment. As we consider programming to be a type of subject requiring intensive and continual training, and as students usually only study on the eve of exams, we thought that the best way for the students to dedicate more study hours to the course was to do more assessments. These assessments have to be well thought out, otherwise we run the risk of not evaluating their true capabilities of programing. Also Parsons [41] suggests that perhaps we are not using correct methods of assessment, possibly we are teaching successfully but assessing badly. Different approaches to assess programming knowledge are also being used with some success [42-44].

From 2011/2012 onwards, teachers decided to make six additional tests corresponding to intermediate topics considered relevant for a gradual understanding of the subsequent topics. Nevertheless, this is a strategy that holds an extra workload for the teachers without any evident improvement from students. Therefore, in this paper we described an assessment methodology applied in 2015/2016 consisting in three tests. These tests were made at key moments, corresponding to topics we consider problematic for the students. We also made changes in the type of questions. We not only considered questions for students to develop the code from scratch but also multiple selection questions, multiple choice questions, true/false questions and assertion reason questions. Our opinion is that it is possible to use a variety of question types to measure the mastery of fundamental concepts, rather than the specific skill of programming. Moreover, we also consider that it is possible to assess different difficulty levels through different types of questions. In this paper, a correlation analyses among the different test/exam questions will be done in order to evaluate the suitability of question types most appropriate for assessing programming. Therefore, this study intends to reach the following goals:

- To determine the correlation of individual questions of distributed assessment with each other and with the final exam;
- To determine which type of questions were correlated with the type of final exam;
- To determine which topics were correlated with the students final exam performance;
- To determine which level of questions best predict students final exam performance;
- To verify if different forms of the same question are equivalent.

The remainder of this paper is structured as follows: section II describes the study design, section III details the study results analysis and section IV ends with some reflections and conclusions about the produced research.

II. THE STUDY DESIGN

Our study focuses on the introductory programming course of the 1st year and 1st semester of Biomedical Engineering degree of the ISEC-IPC (a Portuguese acronym for Coimbra Institute of Engineering of the Polytechnic Institute of Coimbra). This course uses as introductory language the C language and it is followed by a 2nd semester course with C# and object oriented concepts. The subjects covered include the basic concepts taught in an introductory programming course using a procedural programming language, like data types, operators and expressions, standard input and output formatted data, data structures, functions, arrays and string manipulation. In the first three weeks students solve programming problems using sequential, selection and repetitive structures through pseudocode. Only after a certain comprehension of the subject the problems are solved using the C language giving emphasis to the syntax details (Table 1). This course has 4 contact hours per week, 1 hour of theoretic class and 3 hours of lab classes in two groups of about 30 students each (51 students were enrolled in this study), corresponding to 5.5 ECTS. In addition teachers offer 6 more hours per week to clarify student doubts, during the entire semester.

TABLE I. PROGRAMMING LANGUAGE USED AND CONTENTS OF EACH QUESTION IN TESTS AND EXAMS

Question	Language	Contents
Q1T1	Pseudocode	Simple selections
Q2T1	Pseudocode	Simple selections
Q3T1	Pseudocode	Chained selections
Q4T1	Pseudocode	Chained selections
Q1T2	C	Loops
Q2T2	C	Loops
Q3T2	C	Loops
Q1T3	C	Vectors
Q2T3	C	Variable types
Q3T3	C	Functions
Q1Ex	C	Selections, loops, vectors
Q2Ex	C	Strings

As we consider this type of subject requiring a special approach implying an on-going monitoring we decided to have 3 tests throughout the year including different types of questions, not only implying the writing of code from scratch. In the set of questions we tried to follow some of the recommendations of how to write an introductory programming test [45]: to keep the questions as simple as possible; simplifying its preambles as much as possible; to ensure that students are familiar with the type of questions; to avoid variable names that are easily confused with one another or with other symbols; to include questions with different levels of difficulties; to include some multiple-choice questions; to include some code-reading questions; to include questions of different forms.

However, in the final exam we considered an approach, where students have to make programs interconnecting different topics, similar to what happens in the majority of programming subjects. In order to pass, a student should have the principles of computational thinking. The student needs to know how to conceptualize a solution, translate it algorithmically and implement it in a programming language. We consider that all engineering degree students should demonstrate computational thinking abilities in order to pass an introductory programming course.

In addition to these assessment elements, we still use another element; we used an on-line game that was introduced to the students in the first practical class, which, to our knowledge, has never been used with this objective in an introductory programming course. This game is introduced to the students in the first class. It is a type of angry bird game with underlying programming logic, the Code Studio game (https://studio.code.org/hoc/1). We consider that to have, from an early stage, some idea of the existing computational thinking skills of the students is very important to allow teachers to focus more in strategies to help students to acquire this computational thinking. Predicting the success of students participating in introductory programming courses has been an active research area for several years. Until recently, no variables or tests have had any significant predictive power. In

another study [46] we decided to use a test developed by Dehnadi and Bornat at Middlesex University [47], but the results were not conclusive. This game consists in dragging blocks of code that when executed are translated by visual effects in the game. The first activities are purely sequential but, as the game progresses, there is the introduction of repetitive structures, unmovable fixed blocks (structures) or structures with single or chained selections. All students undergoing this game managed to achieve all the 20 levels. However, the number of attempts and lines of code to reach the top level varied widely (from 82 to 202 lines)

The first test, all in pseudocode, involved questions about the topic selections. It included four questions. The 1st question (intermediate (I) level and of type T/F - Q1T1) involved the analysis of a pseudocode program on simple selections. The students had to analyse a set of statements about the presented program, marking each one as True or False. In the 2nd question (basic (B) level where the students had to do a complete code - Q2T1) students had to perform a very simple program in pseudocode also on the topic of simple selections. The 3rd and 4th questions (advanced (A) level with the same set of questions but indicated in different formats (T/F – Q4T1 and prediction of output (PoO) - Q3T1)) involved the analysis of a more complicated program on the subject of chained selections. Students had to answer the 3rd question indicating the expected output for various input data. In the 4th question exactly the same aspects of question 3 were evaluated but now the students had to give their answers in the form of T/F. Although this entire test was done in pseudocode, we hesitated in doing it, because there is no tool in which the students can test pseudocode. However, we did not want to risk further delay waiting for students to acquire C language as it would not have been beneficial to them.

The 2nd test involved questions about loops, all in C language. It included three questions. The 1st question (intermediate level of Multiple Choice Question (MCQ) type - Q1T2) involved the analysis of various encodings including coded loops with the `for` and `while` structures. Students had to indicate, when executed, which could generate the displayed output. In the 2nd question (another intermediate level to code - Q2T2) students had to perform a program in C language also focusing on the topic of loops. In the 3rd question (an advanced level of the type of Single Choice Question (SCQ) - Q3T2) the students had to analyse a coded C program with chained loops and select which of the presented outputs would be generated by the implementation of this program.

The 3rd test involved questions on the latest studied topics: vectors and functions. In the 1st question (basic level to code - Q1T3) students had to perform a program in C language focusing on the topic of vectors. In the 2nd question (of advanced level to predict the output - Q2T3) the students had to predict the output of running a presented program involving concepts of local, global and static variables. This question was removed from this study due to an unexpected last minute change, which was considered confusing for students. In question 3 (intermediate level to "fill in the blanks" (FiB) with code - Q3T3), students had to complete a coding, which included definition and call of functions.

TABLE II. LEVEL AND QUESTION TYPE FOR EACH QUESTION OF THE TESTS AND EXAMS

Questions	Level			Question type					
	B	*I*	*A*	*T/F*	*SCQ*	*MCQ*	*Code*	*FiB*	*PoO*
Q1T1		X		X					
Q2T1	X						X		
Q3T1			X						X
Q4T1			X	X					
Q1T2		X				X			
Q2T2		X					X		
Q3T2			X		X				
Q1T3	X						X		
Q2T3			X						X
Q3T3		X						X	
Q1Ex			X				X		
Q2Ex			X				X		

Note that in order to ensure that students did not respond at random or by guessing, each answer of T/F, SCQ or MCQ had penalties for each response provided wrong.

The final exam included two questions where students had to write complete programs in C language. The first one was a generic question where students had to apply concepts as selections, loops and vectors. For us, this question was an exercise that thoroughly evaluated the expected computational thinking in an introductory programming course. Question 2 focused specifically on strings.

III. STUDY RESULTS ANALYSIS

In regards to the obtained marks, we can make the following considerations, whose obtained statistics are shown in Table 3.

• In test1 there was no significant difference between the averages of the obtained marks in each of the questions. Despite the minimum marks obtained in question 1 and 2 being very distinct in relation to the ones obtained in questions 3 and 4, these results do not mean exactly what they seem. These minimums were obtained by only two students in question 1 and by one student in question 2. These students were asked about this situation, since it did not make sense for us to have high rankings in similar questions but with a higher degree of difficulty. Students said they were wrong in the selection of their answers. As each wrong answer had strong penalties associated, these options reflected negatively on the final result obtained in this question.

• Significant differences between the average scores, obtained in Question 1 and the other two questions, were found in the 2nd test. Question 3 was considered difficult at the moment it was given. It involved a code with chained loops and the students had to know how to predict the output and then chose a single answer. However, the outputs presented as answers were very similar making it harder to come to a conclusion. Thus only proficient students in this topic would be able to see the subtlety of encoding. Questions 1 (MCQ) and 2 (make code) were of intermediate level, although the 1st still had more complex algorithmic aspects. A significant difference in

marks may be due to question format. We believe that in this phase of C knowledge the MCQ format was much more appropriate. The goal was to see if the students had a complete understanding about operating loops, regardless of structure used (`while` or `for`). At this stage, asking students to make a complete C program with all syntactical details associated revelled a difficult and discouraging task. Additionally, students had no complete code in this test to guide them on all the necessary syntactical details. So, we believe that these are the factors responsible for the low average marks in question 2. We believe that this difference was not observed between question 1 and 2 results of test1 precisely for this reason. In an early stage of programming learning, students need a template to guide them on all the syntactical *minutiae* necessary for the proper functioning of a program. The `scanf` and `printf` instructions, including libraries, among other things, are significant challenges in this stage and are not in our view synonyms of programming skills.

In our opinion, this is further evidence on the suitability of certain formats of questions for programming competence assessments.

• In the 3rd test there was no significant difference between the averages obtained in the marks of the several questions, however, there have been quite a few variations between the classifications, with a clear distinction between students with good marks and those with many difficulties. We think that these results are explained by the fact that at this stage there is clear distinction between the students that will get higher and lower grades in programming assessments. The students with higher grades obtained good results in the various types of exercises regardless of the question types or topics covered format.

TABLE III. STATISTICS

	Test1				Test2			Test3			Exam		
	Q1T1	**Q2T1**	**Q3T1**	**Q4T1**	**Q1T2**	**Q2T2**	**Q3T2**	**Q1T3**	**Q2T3**	**Q3T3**	**Q1Ex**	**Q2Ex**	**TotEx**
Average	92.24	92.55	97.55	97.55	80.88	49.75	58.82	65.15	68.98	54.92	42.74	34.93	11.30
Minimum	16.67	15.00	58.33	75.00	0.00	0.00	0.00	0.00	0.00	5.00	2.50	0.00	4.72
Maximum	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	90.00	85.00	18.63
Std. Dev.	18.45	17.50	8.19	6.43	26.45	30.59	52.54	29.25	35.32	31.05	23.53	22.45	3.55
N	**51**	**51**	**51**	**51**	**51**	**50**	**51**	**49**	**49**	**49**	**49**	**44**	**44**

Table 4 shows the correlations among the questions of Test1.

TABLE IV. CORRELATIONS AMONG THE QUESTIONS OF THE TEST 1

	Q1T1			
Q2T1	------	**Q2T1**		
Q3T1	.912**	------	**Q3T1**	
Q4T1	.917**	------	.976**	**Q4T1**
Q1Ex	.347*	------	------	------
Q2Ex	------	------	------	------
TotEx	.338*	------	------	------

The 1st test analysis revealed the following:

• In Q1T1 (T/F question): The majority of students obtained good results in this question. It was about simple selections. There were correlations between the results obtained in this question (Q1T1) and the results obtained in question 1 of the exam (Q1Ex) and also with the total result of the final exam (TotEx). Question 1 of the exam (Q1Ex) was to perform code and the type of question 1 of test1 (Q1T1) was of T/F type. Therefore, we can say that the students who got better ratings in T/F question were also those who have better ratings on a question that implied to make code in the final exam.

• In Q2T1 (question to code): Most students did this exercise well, getting high rankings. There were no correlations between the results obtained in this question and other question results of the same test or with any of the questions of the final exam. At first sight it could seem curious because this question implied the writing of code; which was also present in the final exam questions. However, we think that students had a model (in question 1 of test1) of code they could follow as an example to make the codification of question 2. There are lots of details in a complete program that a student must master to have a good mark and that does not mean mastery in computer programming and in the underlying algorithms. Therefore, we can conclude that this type of question was not important to use in this stage.

• In Q3T1 and Q4T1: There was a correlation between the results obtained in these 2 questions and also between each of them and question 1 of test1 (Q1T1). Moreover, further analysis revealed that students did not randomly answer Q3T1 and Q4T1 questions. This is because these questions evaluated the same type of knowledge but were intentionally placed in two different ways (Q3T1 involved the indication of the output of the presented code and Q4T1 in the format of T/F), checking the coincidence of correct results with a true understanding. We also wanted to verify if different forms of the same question are equivalent.

To conclude, test1 analysis concerning the different types of questions we found that:

• The T/F question (existing on Q1T1) had correlation with question 1 of the exam (Q1Ex), which was making code and

with the final results of the exam. However, this type of correlation did not occur with another T/F question (Q4T1). We did not find justification for this; especially do to the fact that this question was significantly more difficult than the first one (Q1T1) and students had reached even better results than in the first one (Q1T1). However, we can make some observations about this. This was an issue discussed with the students. These indicated that they realized that the weight of these questions (Q4T1) and Q3T1 would be considerably higher than Q1T1. While Q1T1 had only 6 options to choose Q3T1 had 9 and Q4T1 had 10. Thus, students concentrated much more in these two questions. Additionally, they had to think in the same question in two different ways and many of them said they automatically excluded some choices to a question by a particular existing reflection in the other. When students are aware that each question has a different value this can influence the time a student dedicates to solving the questions. Especially, if the question has been seen before, the student had the opportunity to think about it making it easier to answer it correctly.

• The type of question to "indicate the output" (existing in Q3T1) had correlation with the types of question T/F (existing on Q1T1 and Q1T4). However, no type of correlation was obtained with this type of question and question 1 of the exam (Q1Ex) or with this type of question and the final result of the exam (TotEx).

Table 5 shows the correlations with the questions of the 2nd Test. The analysis of the this test revealed the following:

• In Q1T2 (MCQ) - Most students seemed to understand the concepts of simple loops well, as the average results obtained in this question were high. No difference was noticed in understanding between the loops involving the structures `while` or `for`. There were correlations between the results obtained in this question and the results obtained in question 1, 3 and 4 of test1 (Q1T1, Q3T1 and Q4T1). We think that this happened because all these questions involved true understanding of concepts putting aside the details of a complete program.

TABLE V. CORRELATIONS WITH THE QUESTIONS OF THE 2ND TEST

	Q1T1	Q2T1	Q3T1	Q4T1	Q1Ex	Q2Ex	TotEx		
Q1T2	.812**		.831**	.824**	------	------	------	**Q1T2**	
Q2T2	------	------	------	------	------	------	------	------	Q2T2
Q3T2	.363**	------	.369**	.361**	------	------	------	.465**	------

• In Q2T2 (Make code): Most of the students struggled to make the code of this question, resulting in low marks. There were no correlations between the results obtained in this question and the final results obtained in the programming exam. The rationale may be related to the fact that it has been the question with lower ratings, due to the motive explained above.

• In Q3T2 (SCQ) – It seemed that most students did not understand chained loops as the average marks obtained in this question were relatively low (58.82%). There were correlations between the results obtained in this question (Q3T2) and the results obtained in questions 1, 3 and 4 of test1 (Q1T1, Q3T1 and Q4T1). There were correlations between question Q3T2 and question 1 of the same test (Q1T2). Perhaps, a connection among MCQ, SCQ and T/F questions could be established. However, no correlations were found between the results obtained in this question and the results obtained in any of the final exam questions or with the global final exam. However, intuitive analysis showed that students who had better marks on this question generally obtained better final results in the programming exam. Perhaps the small number of students who obtained positive or good marks in this question made it impossible to obtain the expected correlations. This was a question analysis considered difficult in this period, involving more elaborate aspects of abstraction than the final examination questions.

The 2nd test was considered the most difficult by the majority of students and it was the one in which the marks lowered the most, perhaps for that reason it was the only one with no correlation with the various partial and total results of the final exam. The type of topics involved was also not so intensified in the final exam. Therefore, it is possible that a different incidence in the different topics made this difference.

Table 6 shows the correlations with the questions of Test3.

TABLE VI. CORRELATIONS WITH THE QUESTIONS OF THE 3RD TEST

	Q1T1	Q2T1	Q3T1	Q4T1	Q1T2	Q2T2	Q3T2	Q1Ex	Q2Ex	TotEx		
Q1T3	.344*	------	------	------	------	------	------	.722**	.606**	.753**	**Q1T3**	
Q3T3	.425**	------	------	------	------	------	------	.749**	.601**	.722**	.753**	.563**

Test3 analysis revealed the following:

• In Q1T3 (Make code): The average of the marks obtained in this question seem to show that most students did this

exercise well. However, an average only by itself says very little. It was verified that there were students with very good grades and students with bad grades. There were correlations between the results obtained in this question and the results obtained in question1 of test1 (Q1T1) and questions 1 and 2 of the exam (Q1Ex and Q2Ex) and consequently with the final results of the exam (TotEx). These correlations indicate, for us, that the aptitude for programming could be already foreseen with this test.

• In Q3T3 (complete code): There were correlations between the results obtained in this question and the results obtained in question 1 of test1 (Q1T1) and question 1 and 2 of the exam (Q1Ex and Q2Ex) and consequently with the final result of the exam (TotEx). These results also confirm our assumption about the tendency students show about their programming performance in this stage.

In this test a correlation between all the questions (Q1T3, Q2T3 and Q3T3) and question 1 of test1 (Q1T1) (T/F) was obtained. Correlations between all the questions of this test with each other (Q1xQ2, Q1XQ3, Q2xQ3) were also obtained. In our perspective this means that whoever revealed programming knowledge at this stage could do so on different topics (vectors and functions) and also on different question formats.

Finally, we can say that, the strong correlations obtained with the 1st test show us the students' capacity for programming from an early stage. The trend is revealed from the start and regardless of question format! Regarding the final exam: With question 1 of the final exam (Q1Ex) and the entire final exam (TotEx) only correlations with question 1 of test1 (Q1T1) and with all the questions of test3 (Q1T3, Q2T3 and Q3T3) were found. Question 1 of test1 revealed to us a tendency of aptitude for programming, right from the start. We expected a similar correlation with Q4T1 because they are of the same type, but it was not the case, probably due to the reason already mentioned.

Test3 was carried out at a stage where only the students with high grades could already follow the matter. But test2's questions showed no correlation with the final exam questions. In our perspective, this happened because it was a more difficult test, about a topic (loops) explored in depth, where only students with high grades excelled, showing a great difference in marks. The number of students who achieved better ratings was not sufficient to obtain a correlation, so we cannot make other observations.

In our analysis question 2 of the exam (Q2Ex) was not considered because it had an exceptional character. That is, this question focused on a very specific matter (strings) not evaluated in the tests. However, the correlations between this question and some test3 questions may mean that the students with high grades also had better ratings in this question.

Regarding the analysis of students playing “Code.Studio”, all students were able to complete all the included levels. However, what the platform allows to register is the number of lines of code and not the number of attempts to reach each level. Many students only managed to achieve success after several trial and error attempts. However, the number of lines of code reached gave us some guidelines. Although no correlations with statistical significance were obtained, a negative trend was observed between the number of lines of code and each of the test questions and exam. This correlation indicated that students who scored higher in each of the tests and the exam questions were also those who performed the activity with the lowest number of lines of code which indicates a structured thinking of students.

IV. Reflexion and Conclusions

The problems associated with programming teaching and learning are not new but are still far from being solved. The causes are identified but the approaches to follow can be numerous and there is still no consensus on the best one and the most convenient. Also, from the viewpoint of the student, there is evidence of the best approaches to follow in order to obtain good results [40]. However, the pedagogical strategies used by the teacher will have a key role here. An approach involving a closer teaching and monitoring of students has given evidence of this effectiveness [48]. This strategy cannot be disassociated from a desirable continuous evaluation. However, with no means and resources for a continuous evaluation we consider that at least one distributed evaluation can bring forth very useful indicators. In this sense, this paper is somewhat of an alert. The existence of intermediate evaluations on key topics considered fundamental is a crucial element. This is a technique that many teachers avoid due to the number of students usually involved and the extra work that is involved. Additionally, teachers feel that these efforts have no turnover in terms of student learning. However, there are formats of questions as MCQ, T/F or "fill in the blanks" which in our opinion could also assess all the levels of an educational objectives taxonomy for any topic.

In our view a contribution to solve this problem will intensify especially assessments with relevant and probing questions that are easy to correct in order to determine the progression of student knowledge. Continuous assessment is useful for both students and teachers. For students it helps them to understand their own learning progression and difficulties. For teachers it enables them to see the students learning progression and eventually to give personalized support towards the demonstrated doubts and adjust the teaching materials accordingly.

We think that the importance is not to make the distinction between students who pass or fail the introductory programming subject; because we could be evaluating it in a wrong way. Also many authors suggest that many students who pass in programming don’t know how to program well [49-51]. We are interested in making assessments that distinguish between those students who developed the correct programming ability and those who did not develop it. We think that in an introductory programming course students do not need to know certain theoretical principles and abstractions in great detail, but at the end they should show the ability to generate a program in which the computational thinking is underlying and know how to solve a programming problem in its entirety.

As to our research questions we can state that, besides the individual correlations in some questions of some tests already explained there were correlations between the questions Q1T1 (T/F, of intermediate level), Q1T3 (To code, basic level), Q3T3 (fill in blanks, of intermediate level) and the final exam.

With reference to the topics, the loop topic does not seem to be important for the final programming performance, perhaps because the questions in the final exam were not really related with this topic, at least in such depth as the one evaluated in the tests.

We also verified that different forms of the same question are equivalent; therefore they could be used to assess equal knowledge. This happened between Q3T1 and Q4T1, so we suggest using a variety of formats, the ones the teacher consider more convenient, because they could also evaluate the intended knowledge well.

In conclusion, we think that in order for a student to pass in an introductory computer-programming course they will have to have the ability to solve a computational problem in its entirety. The student should master the competences of abstraction, reasoning and algorithmic thinking at an acceptable level. However, until they master these competences a variety of question types could be used instead of free code text. During this phase we want students to master each of the particular topics in some depth. Therefore, the level of demand for each exercise should neither be too demanding nor to easy, but have an intermediate level of difficulty to motivate and challenge students. So it is important that students start to master each particular topic in some depth and gradually get to know how to integrate each of the topics with each other in order to be able to make a program in its entirety. We also verified that from an early stage it is possible to predict the student ability to program and be attentive to the most problematic cases, using a variety of question formats without being laborious for the teacher.

Finally, it is important to state that at the end of the semester an inquiry was produced to evaluate the students' opinions about this type of evaluation concepts and all the students referred that they felt more motivated in performing multiple choice questions or even code completion questions instead of developing a full program, which can constitute an important direction to be analysed in future works.

ACKNOWLEDGMENT

The authors would like to thank FLAD (a Portuguese Acronym for Fundação Luso-Americana para o Desenvolvimento), ISEC-IPC and CISUC for all the support in the implementation of the work, publication and presentation of the paper and to all students that participated in the experiment.

REFERENCES

[1] R. Lister, "On blooming first year programming, and its blooming assessment," in *Proc. of the Australasian Conference on Computing Education,* Melbourne, Australia, New York, USA, 4-6 December 2000, pp.158-162.

[2] S. Bergin and R. Reilly, "The influence of motivation and comfort level on learning to program," in *Proc. of 17th Annual Workshop on the Psychology of Programming Interest Group*, Sussex, UK, 29th June – 1st July 2005, pp 293-304.

[3] R. Lister, B. Simon, E. Thompson, J. L. Whalley and C. Prasad, "Not seeing the forest for the trees: novice programmers and the SOLO taxonomy", *SIGCSE Bulleti*n, vol. 38, no. 3, pp. 118-122, 2006.

[4] G. Gonzalez, "A systematic approach to active and cooperative learning in CS1 and its effects on CS2," in *Proc of 37th SIGCSE Technical Symposium on Computer Science Education*, Houston, USA, March 2006, pp. 133-137.

[5] E. W. Dijkstra, "On the Cruelty of Really Teaching Computing Science", *Communications of the ACM*, vol. 32, no. 12, pp. 1388-1404, 1989.

[6] D. Lee, M. Rodrigo, R. Baker, J. Sugay and A. Coronel, "Exploring the relationship between novice programmer confusion and achievement," in *Proc. of the 4th international conference on Affective Computing and Intelligent Interaction (ACII'11)*, Memphis, TN, USA, 2011, pp. 175-184.

[7] E. Lahtinen, K. Ala-Mutka and H.-M. Järvinen, "A study of difficulties of novice programmers," in *Proc. of 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, Caparica, Portugal, June 27-29 2005, pp. 14-18.

[8] T. Jenkins, "On the difficulty of learning to program," in *Proc. of 3rd Annual LTSN Information and Computer Science Conference*, Loughborough, UK, August 2002, pp. 53-57.

[9] A. Carbone, J., Ceddia, D'Souza, Simon and R. Mason, "Student Concerns in Introductory Programming Courses," in *Proc. of the fifteenth Australasian Computing Education conference (ACE'13)*, Adelaide, Australia, 2013, pp. 41-50.

[10] A. Gomes and A. J., Mendes, "Learning to program - difficulties and solutions," in *Proc. of the International Conference on Engineering Education*, Coimbra, Portugal, 2007, CD-ROM.

[11] H. Krushkov, M. Krushkova, M., V. Atanasov and M. Krushkova, "A Computer –Based Tutoring System for Programming", *Mathematics and Mathematical Education* (in Bulgarian), 2009.

[12] G. Rößling, "A Family of tools for supporting the learning of programming", *Algorithms*, vol. 3, pp. 168-182.

[13] E. Verdú, L. Regueras, M. Verdú, J. Leal, J. and J. Castro, J., "A distributed system for learning programming online", *Computers & Education*, vol. 58, no. 1, pp. 1-10, 2012.

[14] S. Salleha, Z., Shukura and H. Judib, "Analysis of Research in Programming Teaching Tools: An Initial Review," in *Proc. of 13th International Educational Technology Conference*, Kuala Lumpur, Malasya, 2013, pp. 127-135

[15] A. Santos, A. Gomes, and A. J. Mendes, "Integrating New Technologies and Existing Tools to Promote Programming Learning," *Algorithms*, vol. 3, no. 2, pp. 183-196, 2010.

[16] C. Rosiene and J. Rosiene, "Flipping a Programming Course: the Good, the Bad, and the Ugly," in *Proc of the ASEE/IEEE Frontiers in Education Conference Proceeding*, El Paso, TX, USA, October 2015, pp. 803-805.

[17] W. A. Kunkle and R. B. Allen, "The Impact of Different Teaching Approaches and Languages on Student Learning of Introductory Programming Concepts," *ACM Transactions on Computing Education*, vol. 16, no. 1, art. 3, pp. 1-26, 2016.

[18] V. Isomottonen and V. Tirronen, "Teaching Programming by Emphasizing Self-Direction: How Did Students React to the Active Role Required of Them?", *ACM Transactions on Computing Education*, vol. 13, no. 2, 2013.

[19] A. Vihavainen, M. Paksula and M. Luukkainen, "Extreme apprenticeship method in teaching programming for beginners," in *Proc of the 42nd ACM technical symposium on Computer science education (SIGCSE '11)*, Dallas, TX, USA, 2011, pp. 93-98.

[20] Y. Uchida, S. Matsuno, T. Ito, M. Sakamoto, "A proposal for teaching programming through the Five-Step Method," *Journal of Robotics, Networking and Artificial Life*, vol. 2, no. 3, pp. 153-156, 2015.

[21] J. Sajaniemi, M. Kuittinen and T. Tikansalo, "A Study of the Development of Students' Visualizations of Program State during an Elementary Object-Oriented Programming Course," *ACM Journal on*

Educational Resources in Computing, vol. 7, no 4, art. no. 3, January 2008.

[22] A. Pears, S. Seidman, L. Malmi, L. Mannila, E. Adams, J. Bennedsen, M. Devlin and J. Paterson, "A survey of literature on the teaching of introductory programming," *SIGCSE Bulletin*, vol. 39, no 4, pp. 204-223, December 2007.

[23] R. Anderson, B. Simon, S. A. Wolfman, T., VanDeGrift and K. Yasuhara "Experiences with a tablet PC based lecture presentation system in computer science courses", *SIGCSE Bulletin*, vol. 36, no. 1, pp. 56-60, 2004.

[24] J. Brock, R. Bruce and S. Reiser, "Using Arduino for introductory programming courses", *Journal of Computing Sciences in Colleges*, vol. 25, no. 2, pp. 129-139, 2009.

[25] T. Soule and R. Heckendorn, "COTSBots: computationally powerful, low-cost robots for Computer Science curriculums", *Journal of Computing Sciences in Colleges*, vol. 27, no. 1, pp. 180-187, 2011.

[26] I. Greenberg, D. Kumar and D. Xu, "Creative coding and visual portfolios for CS1," in *Proc. of the 43rd ACM technical symposium on Computer Science Education (SIGCSE'12*), Raleigh, North Carolina, USA, 2012, pp. 247-252.

[27] M. McGill, "Learning to program with personal robots: Influences on student motivation", *ACM Transations on Computers Education*, vol. 12, no. 1, art. 4, 2012.

[28] J. Davis, B. Wellman, M. Anderson and M. Raines, "Providing robotic experiences through object-based programming (PREOP)", in *Proc. of the 2009 Alice Symposium*, Durham, North Carolina, USA, pp. 1-5.

[29] T. Lauwers, I. Nourbakhsh and E. Hamner, "CSbots: design and deployment of a robot designed for the CS1 classroom", in *Proc. of the 40th ACM technical symposium on Computer Science Education (SIGCSE'09)*, Chattanooga, Tennessee, USA, 2009.

[30] C. Martin and J. Hughes, "Robot dance: edutainment of engaging learning," in *Proc. of the 23rd Psychology of Programming Interest Group (PPIG'11)*, York, UK, 2009.

[31] W. McWhorter and B. O'Connor, "Do LEGO® Mindstorms® motivate students in CS1?," in *Proc. of the 40th ACM technical symposium on Computer Science Education (SIGCSE'09),* Chattanooga, Tennessee, USA, pp. 438-442.

[32] C. Kazimoglu, M. Kiernan, L. Bacon and L. Mackinnon, "A Serious Game for Developing Computational Thinking and Learning Introductory Computer Programming", *Procedia-Social and Behavioral Journal*, vol. 47, pp. 991–1999, 2012.

[33] G. Linden (2013) Game Maven: Learn to code by writing games. [Online]. Available: http://www.crunchzilla.com/.

[34] M. Eagle and T. Barnes, "Experimental evaluation of an educational game for improved learning in introductory computing," in *Proc. of the 40th ACM technical symposium on Computer Science Education (SIGCSE'09)*, Chattanooga, Tennessee, USA, pp. 321-325, 2009.

[35] K. C. Yeh, "Using an Educational Computer Game as a Motivational Tool for Supplemental Instruction Delivery for Novice Programmers in Learning Computer Programming," in *Proc. of the Society for Information Technology & Teacher Education International,* Charleston, South Carolina, USA, pp. 1611-1616, 2009.

[36] J. M. Rodríguez, A. A. Balcells, A. M. Estévez, G. J. Moreno, M. J. Ferreiro, "A game-based approach to the teaching of object-oriented programming languages", *Computers & Education*, vol. 1, pp. 83-92, 2014.

[37] P. Fotaris, T. Mastoras, R. Leinfellner and R. Yasmine, "From hiscore to high marks: Empirical study of teaching programming through gamification," in *Proc. of the 9th European Conference on Games Based Learning ECGBL*, Steinkjer, Norway, October 2015.

[38] A. Gomes and A. J. Mendes, "Learning to program – difficulties and solutions," in *Proc of the International Conference on Engineering Education,* Coimbra, Portugal, November 2007, CD-ROM.

[39] A. Gomes, and F. Brito Correia, "Errors to avoid in Programming Teaching and Learning," in *Proc. of 7th International Conference of Education, Research and Innovation*, Seville, Spain, November 2014.

[40] A. Gomes, and F. Correia, "The paths taken by good and weak programming students," in *Proc. of E-LEARN 2015 - World Conference on E-Learning*, Kona, Hawaii, U.S.A., October 2015.

[41] D. Parsons, K. Wood and P. Haden, "What Are We Doing When We Assess Programming?" in *Proc. of 17th Australasian Computing Education*, Sidney, Australia, January 2015, pp. 119—127.

[42] D. Clark, "Testing programming skills with multiple choice questions," *Informatics in Education*, vol. 3 no. 2, pp. 161–178, 2004.

[43] J. Hardy, S. Bates, M. Casey, K. Galloway, R. Galloway, A. Kay, A. E., P. Kirsop and H. McQueen, "Student-Generated Content: Enhancing learning through sharing multiple-choice questions," *International Journal of Science Education*, vol. 36, no. 13, pp. 2180–2194, 2014.

[44] S. Azer, "Assessment in a problem-based learning course," *Biochemistry and Molecular Biology Education*, vol. 31, no. 6, pp. 428–43, 2003.

[45] Simon, J. Sheard, D. D'Souza, M. Lopez, A. Luxton-Reilly, I. H. Putro, P. Robbins, D., Teague and J. Whalley, "How (not) to write an introductory programming exam," in *Proc. 17th Australasian Computing Education Conference,* Sydney, Australia, January 2015, pp. 137-146.

[46] A. Gomes and F. Brito Correia, "Programming Education Strategies," in *Proc of 7th International Conference of Education, Research and Innovation*, Seville, Spain, November 2014.

[47] S. Dehnadi, "Testing Programming Aptitude," in *Proc of 18th Workshop of the Psychology of Programming Interest Group*, Sussex, UK, September 2006, pp 22-37.

[48] A. J. Mendes, L. Paquete, A. Cardoso, A. and A. Gomes, "Increasing student commitment in introductory programming learning," in Proc. of the 42th ASEE/IEEE Frontiers in Education Conference – FIE'12, Seattle, USA, 2012, DOI: 10.1109/FIE.2012.6462486

[49] M. Ford and S. Venema, "Assessing the Success of an Introductory Programming Course," *Journal of Information Technology Education*, vol. 9, pp. 133-145, 2010.

[50] L. Thomas, M. Ratcliffe, J. Woodbury and E. Jarman, E., "Learning styles and performance in the introductory programming sequence," *ACM SIGCSE Bulletin*, vol. 34, no. 1, pp. 33-37, 2002.

[51] R. Bornat, S. Dehnadi and Simon, "Mental models, consistency and programming aptitude," *Tenth conference on Australasian Computing Education (ACE '08)*. pp. 53-61, 2008.