Programming education strategies

Conference Paper · January 2014

CITATIONS
4

READS
42

2 authors:

Anabela Gomes
Instituto Politécnico de Coimbra
64 PUBLICATIONS **566** CITATIONS
SEE PROFILE

Fernanda Maria Brito Correia
Instituto Politécnico de Coimbra
20 PUBLICATIONS **28** CITATIONS
SEE PROFILE

PROGRAMMING EDUCATION STRATEGIES

Anabela Gomes [1], Fernanda Brito Correia [1]

[1] Department of Informatics and Systems Engineering (DEIS), Engineering Institute of Coimbra (ISEC), Polytechnic Institute of Coimbra (IPC), Portugal

Abstract

Results in programming courses are often very disappointing mainly when the teachers invest a lot in the subject. There could be a variety of reasons in the origin of this problem. However, the assessment approach used to evaluate students can also have influence. Even if we do not obtain brilliant results, we think it is possible to lead students to obtain the minimum skills required in an initial programming course. This paper shows the application of a methodology that achieved some positive results. Through an approach that required students to do more frequent evaluations, it was possible that students devoted more attention to the programming course with a higher cadence and frequency.

Keywords: Programming teaching and learning, Assessment.

1 INTRODUCTION

The universal problems associated with programming education in many higher education institutions [1-5] leading to high failure levels are well known. Our institution, the Informatics and Systems Department (DEIS) from the Engineering Institute of Coimbra (ISEC) of the Polytechnic Institute of Coimbra (IPC), Portugal, is also facing this problem. Several efforts have been made throughout the world, like the development of tools to assist the teaching/learning of initial programming [11-21]. Although there are reports of some positive results, with the use of some of these tools there are no positive and generalized if one in particular is chosen. Several authors point out different problems, such as programming being naturally difficult, the lack of preparation of students, mainly respecting mathematical skills and lack of abstraction competences, the incorrect study and teaching methods applied to this type of subject, the usage of too complex development environments to meet the pretended needs, among others [6, 7, 8, 9, 10]. In general we agree that all these factors can contribute to failure. However, we think it is our responsibility as teachers of an introductory programming course, in our case the 1st year, 1st semester Biomedical Engineering degree, which is facing the same problems, to try and propose strategies that can change the panorama. It is in this perspective that we present this work focusing indirectly on the study methods and including different assessment strategy.

2 THE STUDY

Our study focuses on the introductory programming course of the 1st year and 1st semester of Biomedical Engineering degree of the ISEC/IPC. This course uses as introductory language the C language and it will be 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. As follows, data types, operators and expressions, standard input and output formatted data, data structures, functions, arrays and string manipulation. In this course we use C language, but in the first three weeks students solve programming problems using sequential, selection and repetitive structures through pseudo code. Only after a certain comprehension of the subject the problems are solved using the C language giving emphasis to the syntax details. This course has 5 contact hours, including 1 hour of theoretic class and 3 hours of lab classes per week in two groups of about 30 students each. In addition teachers offer more 6 hours per week to clarify students' doubts, during the entire semester.

Although failure rates from this course in previous academic years were not very significant they were more than expected when considering the effort that teachers had expended in this course. Thus, in the 2012/2013 academic year we tried a different strategy concerning the assessment method that from the teachers´ perspective could translate into better results. In previous years, students had only one assessment, which took place at the end of the year. In the current year teachers decided to make six additional tests corresponding to intermediate topics considered relevant for a gradual

understanding of the following topics. Although this is a strategy that holds an extra workload for the teachers, they embarked on this because they considered it a way to direct students in devoting themselves to more continuous study and study time towards the course. We consider programming to be a type of subject requiring intensive training, so it was the best way found for the students to dedicate more study hours to the course along the semester.

We also encouraged students to participate as much as possible and expose their doubts. However the number of students per class prevented an effective individualized and permanent student monitoring during class. As we wanted to be as attentive as possible to students that could be future failure cases we used a tool (questionnaire) that would allow us to eventually pay more attention to these students. We used a survey, described in the next section, which according to their authors allows the prediction of future failure cases in programming. Additionally, teachers used a set of strategies to try to improve outcomes in programming learning. Every week teachers were available to support students' difficulties. They had 6 hours per week entirely dedicated to clarify students' doubts. Teachers also told students to feel free to send emails with doubts whenever they wanted. There was also an e-learning platform where teachers uploaded useful content to students according to the topics taught and the progression of the students.

2.1 Questionnaire

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. However we decided to use a test developed by Dehnadi and Bornat at Middlesex University [22]. We administered the questionnaire early in the first class of the course. According to their authors this test enables to classify students according to their consistency in answering a set of similar questions. The overall hypothesis is that only consistent students will be able to learn to program. The test consists in 12 small programs. Each program consists of two or three variable declarations and one, two, or three assignment statements. Fig. 1 shows a sample of it. However we made a small change to each question. We removed the type of variable, which we considered irrelevant for our purpose.

1. Read the following instructions and select the right answer in the columns.
 	
`a = 10;`
`b = 20;`
 	
`a = b;`
	The new values of `a` and `b` are:
	☐ `a = 10` `b = 10`
	☐ `a = 30` `b = 20`
	☐ `a = 0` `b = 10`
	☐ `a = 20` `b = 20`
	☐ `a = 0` `b = 30`
	☐ `a = 10` `b = 20`
	☐ `a = 20` `b = 10`
	☐ `a = 20` `b = 0`
	☐ `a = 10` `b = 30`
	☐ `a = 30` `b = 0`
	Outros valores para `a` e `b`:
	a = **b =**
	a = **b =**
	a = **b =**
		Use this column for drafts.

Fig. 1 Sample of a question of Dehnadi and Bornat test

We wanted further to correlate the results of our questionnaire with the students' marks in the final course tests and with the final programming mark. The idea is to verify the validity of this correlation.

In other words, we wanted to verify if there is a positive correlation between a student's mental model and the student's ability to learn programming. These results are presented in the next section.

2.2 Continuous Assessment Activities

In the 2012/2013 academic year students of the mentioned course could choose between two different modes of assessment. The first mode was similar to the one followed in previous years, consisting only in the final exam. The other second mode consisting in continuous assessment activities including six tests, which covered the topics teachers considered more relevant. These activities corresponded to 50% of the final mark, the other 50% being assessed in the final exam. Students were advised by teachers to follow the second evaluation model. Teachers thought that this model would be better to learn programming and consequently obtain academic success. Tests were spaced in time intervals, coinciding with the temporal moments that teachers considered suitable to focus on important matters for a proper development in the programming learning.

In Test 1 (T1) an encoding in pseudo code was given to students whose aim was to determine the largest of three values. The coding consisted of a set of nested selections. The purpose of the coding was communicated to the student and the student had to point out the truth of each of the 22 sentences marking them as T (True) or F (False). The statements were designed to determine if students completely understood the selections and thereof the explicit and implicit logical conditions.

In Test 2 (T2) coding in C language was provided to the students, which consisted of nested selections. In each selection there was a condition composed of two logical variables. For each one a particular sentence should be printed. The student should indicate the message to be printed when the variables assumed different values. The objective was to verify if the student realized in its fullness the values assumed implicitly and explicitly for each variable in each condition and also the resulting logic condition.

In Test 3 (T3) a set of codifications in C language was given to students. In some cases they should indicate which of the presented encodings produced the desired output. In other cases, they should indicate which of other presented outputs was produced by the given encoding. The coding focused on simple repetitive structures, in order to determine if students understood in its fullness its basic behaviour. Namely, which were the initial condition, the stop condition and the condition that allows the continuation in the repetitive structure as well as their influences in the entire repetitive process.

In Test 4 (T4) students were asked to complete coding in C language, which included mixed selection and repetitive structures. The objective was for the students to complete a program that simulated the calculation of the final scores of a group of students attending a course with a certain number of ranges. The score for each student could be obtained through two alternative modes chosen by students. Each one of the modes included different aspects such as the classification of tests with different weights, the classification of the final exams and students' attendance to classes. The code was given to students completely structured. Students should only complete aspects such as "The selection structure was following all the elements (students) of the repetitive structure", "The condition that checked if the number of student absences was a valid value", "The condition that checked if the number of absences allowed the student to be assessed" or " Simple formulas for the calculation of other statistical expressions".

In Test 5 (T5) students had to answer two questions where for each one they should indicate, which of the presented results would be generated by the presented encodings. Although both encodings included repetitive structures they had a different nature. One had several nested repetitive structures each containing simple expressions. The other had just a simple repetitive structure, but included a call to a function and mathematical expressions of higher complexity.

The Test 6 (T6) focused on elementary aspects of vectors. It included mainly questions implying the use of simple repetitive structures that had to follow and change the values of the vector elements. This was the first test where students had to make codifications. So, the more complex question included the writing of a function receiving as arguments two vectors and its size. The first was a vector of chars and the second a vector of integers. The function should enable the writing, on the screen, of each element of the vector of chars (one character) by repeating this a number of times equal to the integer value stored in the corresponding element of the vector of integers. Each element of the vector of chars should be written on a new line.

3 RESULTS

In this section we present the results achieved by students in the mentioned programming course in the last two academic years, namely 2011/2012 and 2012/2013, using different assessment approaches. Considering all 69 students enrolled in the course, in 2011/2012 the approval rate was 43% and the 68 students enrolled in 2012/2013 the approval rate was 65%. However we can't interpret this information in such a straightforward manner. In 2012/2013 only one student decided not to do the exams while in 2011/2012 there were 16 students that never made any assessment. So in this case, and considering only the students that made at least one assessment we have different values. In the 2011/2012 the approval relative rates were 57% and in the school year 2012/2013 were 66%. However, we are interested in the analyses of the new methodology applied in the 2012/2013 academic year, so our analysis will be concentrated in this academic year.

As teachers we are highly engaged in contributing to the resolution of this problem and so in the next section, a more detailed analysis of the obtained results will be made. It is our intention to weight the new approaches in order to try to continue minimizing this problem.

3.1 Questionnaire Results

The idea of the questionnaire is to predict success or failure even before students have had any contact with any programming language. So we only applied this questionnaire to new students. The questionnaire contains 12 questions similar to the one in Figure 1, giving rise to a 12-tuples describing the mental models applied by a student (e.g. m1, m2, ..., m12)) where mi represents a mental model. The 12-tuples is used to assign each student to one of three categories:

- The consistent group. The students who use the same mental model for most of the questions (regardless of which model).
- The inconsistent group. The students who use several mental models to answer the questions.
- The blank group. The students who refuse to answer the questions.

In the consistent group we have identified 5 different mental models (M1, M2, M5, M10 and M11), which were captured by options in the questionnaire. We tried to cross the students' mental models with the results they obtained in the different assessment components and also with the final global result. However, it was not possible to obtain correlations. From the 68 students, we only got 21 valid tests, not enough to obtain statistical validations. Some students missed the first lesson where this test was done. Some other students made mistakes that forced us to invalidate the tests. However, through an intuitive analysis, we couldn't verify the idea presented by the test author. There were students classified in the consistent group that failed the programming courses. As well, there were students classified in the inconsistent group that not only passed but also had good marks in the course. There could be many reasons for these results and it probably doesn't mean that the test was not valid. It is our intention to continue with this test next year and explore it more deeply.

3.2 Continuous Assessment Results

There were 68 students enrolled in this programming course. Tests T1, T2, T3, T4, T5 and T6 were made by 65, 64, 58, 59, 62 and 55 students, respectively.

The marks averages of each of the tests T1, T2, T3, T4, T5 and T6 were 84.63%, 85.47%, 46.98%, 53.81%, 56.13% and 50.27%, respectively. However, the analysis in terms of average reveals little about the results, so Table 1 presents additional statistical information.

Table1 Statistical information of the six tests

	T1	**T2**	**T3**	**T4**	**T5**	**T6**
Nr of students	65	64	58	59	62	55
Mean	84.63%	85.47%	46.98%	53.81%	56.13%	50.21%
Median	90.00%	90.00%	40.00%	54.00%	50.00%	54.25%

Mode	100.00%	100.00%	40.00%	56.50%	95.00%	55.00%
Std. Dev.	16.163	20.485	23.414	17.322	38.530	25.124
Var.	261.237	419.618	548.193	300.063	1484.565	631.231
Min.	34.00%	10.00%	0.00%	17.00%	0.00%	0.00%
Max	100.00%	100.00%	95.00%	95.00%	100.00%	98.50%

In order to better clarify the results Fig. 2 represents the sequence of graphs corresponding to the marks obtained by each student in each test. In each graph we can observe in the x-axis numbers representing the students and in y-axis the classification obtained by the students in each test.

We can observe that the marks obtained in the two initial tests were very encouraging for sudents. The ones obtained in test T1 were a little better than the ones obtained in test T2 mainly, in our perspective, due to one reason. Even though the type of matter was of the same topic (selection structures) we used more complex conditions in test T2. Although this test was already in C language while the test T1 was in pseudocode we think that this did not influence the results. It was confirmed by students, when asked about the difficulty of the C language interpretation, concerning selection structures, that it was as difficult as pseudocode.

These good results obtained in these two tests can mean several things. In our perspective we believe that the students understood the selection structures and made a proper study. This does not mean that they have acquired all the skills needed to solve problems "from scratch" with the same level of difficulty. However, at that phase, we as teachers were happy with these results. We agree that in a first phase it is important that students understand the concepts, then students should be able to write small code fragments within a well-defined context using these concepts and finally students should be able to produce their own encodings. We believe that programming is an activity that requires time until their complete mastering. So, we agree that in an early stage students should do activities of code interpretation and then gradually students could write small code fragments, evolving up until they can write full programs. However, next year we intend to ask students to start codifying earlier. So, perhaps in the second test instead of having two interpretation questions we should put one interpretation question and the other should involve codifying. Both should be of the same knowledge level. With this, we want to verify if they really understand selection structures.

The sharp fall in the ratings in Test 3 (T3) reveals the difficulties that students have in understanding the repetition structures. Thus we consider that students have not studied enough to understand these concepts. Most students when questioned on the subject stated that they had devoted about the same time to both subjects (selections and repetitions), but only in Test 3 they realized the difficulty of the latter topic (repetition structures).

In this test we noticed a very important aspect especially in the first question when students had to choose which one of the several codifications would generate the presented output. Although all the encodings included repetitive structures they have, for us, different codification difficulty. Some of them use the repetitive structure for while others use the repetitive structure while. We noticed that students understood the codifications using while better than the codifications using for. Perhaps for students the for structure is a very condensed structure. In a unique line of code all three expressions exist. The while is perhaps more “visually” understandable. Another aspect was noted. Some of the codifications were implemented in an increasing manner (the initial value was smaller and increased in each iteration) and others in a decreasing manner (the initial value was bigger and decreased in each iteration). Students found the first type easier. Surprisingly, some students told teachers they weren’t aware that it was possible to have a repetitive structure in a decreasing manner. Teachers questioned themselves if they weren’t able to pass this message. So, next year it is a question that should be stressed. To note that in this test the students had only to analyse code and they did not need to do any coding yet. We believed that if we proposed students activities consisting in the complete write of a program, with this type of topic, certainly they would not have yet a mental structure allowing them to structure the whole program, what could demotivate them more.

Even though the students had to codify in test T4, the test was structured to minimize the algorithmic and codifying procedure. We consider that the results obtained in that test were mainly due to the big

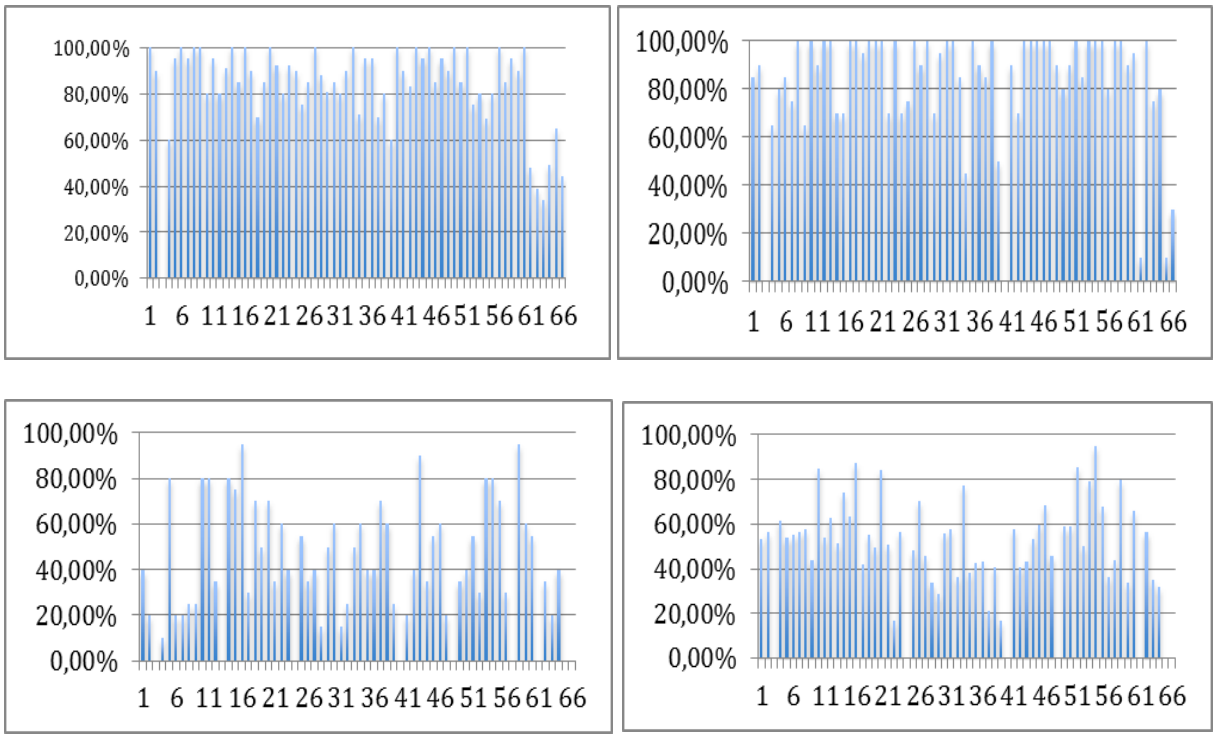
amount of mathematics formula involved. Students reveal difficulty in presenting even basic calculus to solve elementary problems.

Test T5 contained a question with multiple nested repetitive structures each containing simple expressions. The other question was just a simple repetitive structure, but included a call to a function and contained mathematical expressions of a higher complexity. To note that, in these questions, the students hadn't to do any codification but only to analyse code. Some students obtained better results in one question and others in another. However, the differences were not significant nor could we find any pattern among the answers of these two types of questions. Many students revealed many difficulties in both questions, which can be seen in the 6th graph of Figure 2 by the low results obtained. This test also contained another question but we considered it as not being important for this analysis. It consisted only in the filling of blank spaces corresponding to the input parameters and return value of a function.

Test T6 focused on elementary aspects of vectors. The more complicated situations were simple repetitive structures that had to follow and change the elements of the vectors. Despite the general classifications of this test not being considered good, students considered this type of topic easily understandable. However, they stated they had not studied due to having another test that week. This aspect reflects the insufficient and inadequate study methodologies. The majority of students only study when they have assessments, instead of doing it continuously. It also appears that students lack maturity. They miss a lot of lectures (without attendance mandatory) but they don't miss the labs (with two thirds attendance mandatory). They also don't attend the office hours on a regular bases except when close to tests or exam dates.

We can conclude that the results between tests T3, T4, T6 and the final exam are strongly correlated. Thus, students who get better marks in test T3, also have better marks in tests T4, T6 and exam. As already mentioned, the test T5 is a mysterious exception. We also consider that the mastering of the topics included in the test t3 forward reveal the tendency of students to get successful programming results.

Obviously these tests only served to guarantee a minimum follow-up of the matter. To succeed in the final examination, students not only had to make more complex encodings, dealing particularly with matrices, but also involving other topics as strings. Our initial goal was also to include these topics in the tests. However, the topic of strings was only taught in the last 2 weeks of classes. In this period the students had other tests and as such, we did not want to overload them. However, students were alerted to the fact that strings were a very important topic to which they should devote many hours of study outside lessons. We want to stress that the type of the programming content is not so deeply explored as it was in a computer science degree. For instance topics such as pointers, structures, files or linked lists are not taught. The programming course in analysis belongs to a Biomedical Engineering degree.



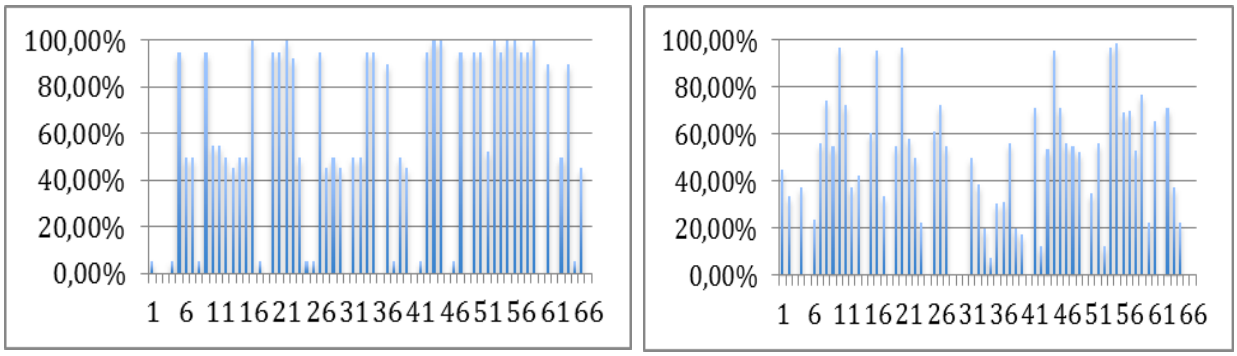


Fig. 2 Students marks in each of the six tests (left to right and up to down)

As already mentioned the relative approval rates were not very different in 2011/2012 and 2012/2013. The same happened with the median marks of the approved students, the majority of these students obtained a final classification around 10 values in both years. We expected better results in the 2012/2013 academic year, however, we seem to understand the underlying reason. Even though most students managed to get positive in the six tests, the global marks obtained in all the tests were relatively low. The same happened with the marks obtained in the exam. Concerning the tests the reasons were already discussed. Concerning the exams we considered mainly two aspects. On one hand, students do not put much effort because they thought the degree of difficulty of the exam was similar to the one used in the tests. On the other hand the final exam had 2 question one about strings and the other on bi-dimensional arrays. Although these topics are more complex we also think that these students did not study and practice these topics adequately. Perhaps if these topics had been involved in a previous evaluation they would have had better results. It is also our intention to include these topics in the interspersed tests next year.

However, our intention is to concentrate on the results obtained in the academic year 2012/2013. The following graphs (Fig. 3, Fig. 4 and Fig. 5) enable us to compare, for each student, the mean of the results in the total of the six tests, with their results in the final exam and the final result, respectively. To note that the total of six tests are classified to 10 values and the exam is also classified to 10 values, so that the total mark is classified on a scale from 0 to 20 (used in Portugal). So the information on x-axis of each graph is presented accordingly.

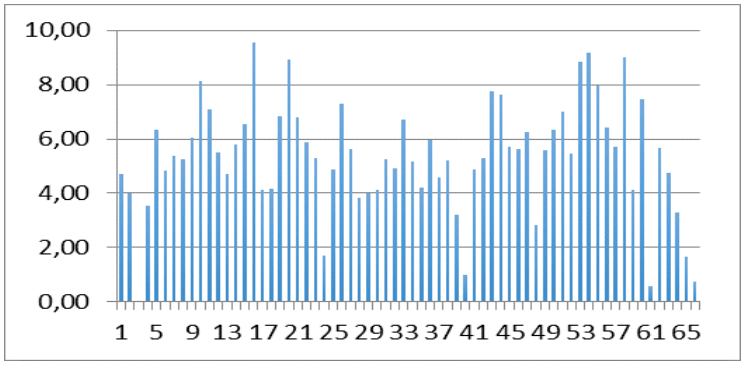


Fig. 3 Median of the six Tests, for each student

We can observe that, as was expected, the results obtained in the final exam were the worsts, what subsequently influenced the final result. The correspondent statistic can be observed in the Table 2.

To get a better and consistent analysis, considering the importance of our methodology, we also crossed the marks obtained in each of the tests with the final grades obtained in programming. This information is presented in Table 2.

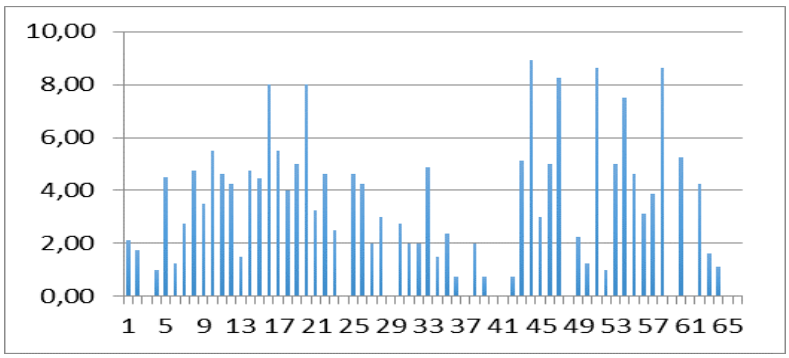


Fig. 4 Classification of the final exam, for each student

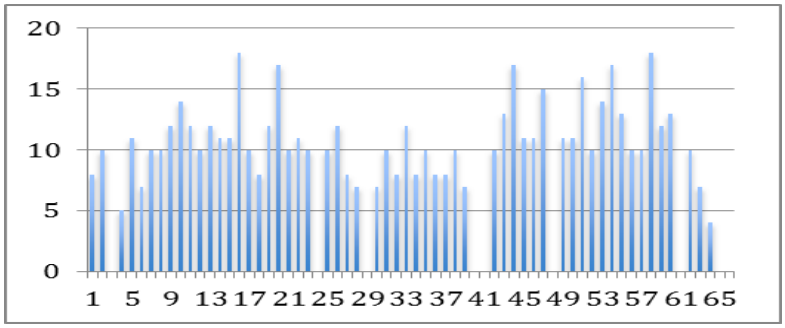


Fig. 5 Final marks for each student

Table 2 Statistical information for each assessment part

	Tests (0-10)	**Final Exam (0-10)**	**Final Grades (0-20)**
Nr of students	65	57	
Mean	5.43	4.68	10.83
Median	5.45	4.63	10.00
Mode	4.11	4.25	10.00
Std. Dev.	1.965	1.857	3.065
Var.	3.861	3.451	9.397
Min.	0.58	1.00	4.00
Max	9.56	8.90	18.00

To get a better and consistent analysis, considering the importance of our methodology, we also crossed the marks obtained in each of the tests with the final grades obtained in programming. This information is presented in Table 3.

The correlation analysis enabled us to extract the following conclusions. Students who obtained the best marks in the exams were also the ones who obtained the best marks in all the tests. Consequently, students who obtained better final results were also the ones who obtained better results in partial tests. There were strong correlations between the scores of all the tests and the exam scores, except with test 5 (T5). In this test, we consider that there were essentially three types of

classifications corresponding to three types of answers. Twenty-six students answered correctly almost all the test (the two relevant questions), nineteen students answered correctly nearly half of the test (one relevant question) and ten students did not answer correctly any question. Even though we didn´t find any strong justification we think that it was a strange test for the students perspective. There was a big variation in the results. The majority of students obtained 95% but there were also students with 0%. However there are some questions to answer. Had the level of difficulty increased more than some students expected? Were some students satisfied with previous results thinking they can study less to obtain satisfactory results? Was the workload of the other courses increased not letting time to students dedicate to that course? After all we consider that, even though the overall classifications (tests and final exam) were not high, the realization of all these tests had a positive influence in the exams results and consequently in the final classification.

Table 3 Person correlation between the tests marks and the exam marks (**at 0,01 level (2-tailed))

	Exam marks
T1 marks	.369**
T2 marks	.372**
T3 marks	.383**
T4 marks	.459**
T5 marks	X
T6 marks	.461**

4 CONCLUSION

The poor results associated with a programing course in analyses led teachers to change the assessment methodology. In this paper the authors compare two different assessment methodologies applied in the two academic years of 2011/2012 and 2012/2013. The global results obtained in the academic years of comparison had been not very different, however a small increasing in the success rate was verified. It is our intention, not to give up using the strategies that we believe may help to minimize the problem. In that way, in the current school year we will follow a similar assessment methodology but with some specific strategies to engage the students in subjects in which they show more difficulties. In particular, we intend to conduct analysis of errors made by students supplemented with individual interviews in order to better understand the causes of these errors and also to individually motivate these students to improve their results.

REFERENCES

[1] Jenkins, T. “On the difficulty of learning to program.” In the 3rd Annual LTSN-ICS. 22-27 August 2002. *Proceedings of the 3rd Annual LTSN-ICS.* Loughborough University, United Kingdom, p. 53-58.

[2] Lahtinen, E., Ala-Mutka, K. and Järvinen, H. “A study of difficulties of novice programmers.” In the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education. 27-29 June 2005. *Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*. Monte de Caparica, Portugal, p. 64-68.

[3] Lister, R., Simon, B., Thompson, E., Whalley, J. L. and Prasad, C. September 2006. “Not seeing the forest for the trees: novice programmers and the SOLO taxonomy.” *SIGCSE Bulletin*, Vol. 38 (3), pp. 118-122.

[4] Bruce, C. S. and McMahon, C. A. 2002. *Contemporary Developments in Teaching and Learning Introductory Programming: Towards a Research Proposal*. Teaching and Learning Report. Faculty of Information Technology, Queensland University of Technology, Brisbane, Australia.

[5] Lister, R. “On blooming first year programming, and its blooming assessment.” In the Australasian Conference on Computing Education. 4-6 December 2000. *Proceedings of the Australasian Conference on Computing Education.* Melbourne, Australia. ACM New York, NY, USA, p.158-162.

[6] Gray, W. D., Goldberg, N. C. and Byrnes, S. A. June 2007. “Novices and programming: Merely a difficult subject (why?) or a means to mastering metacognitive skills? [Review of the book Studying the Novice Programmer]”. *Journal of Educational Research on Computers*, Vol. 9 (1), pp. 131-140.

[7] Dijkstra, E. W. December 1989. “On the Cruelty of Really Teaching Computing Science.” *Communications of the ACM*, Vol. 32 (12), pp. 1388-1404.

[8] Byrne, P. and Lyons, G. September 2001. “The effect of student attributes on success in programming.” *SIGCSE Bulletin*, Vol. 33 (3), pp. 49-52.

[9] Martins, S., Mendes, A.J. e Figueiredo, A.D. “Student reflexions as an influence in the dynamics of an introductory programming course.” In the 41st Annual Frontiers in Education Conference. 12-15 October de 2011. *Proceedings of the 41st Annual Frontiers in Education Conference.* Rapid City, USA, pp. T1A1-T1A6.

[10] Bennedsen, J.; Caspersen, M. Abstraction ability as an indicator of success for learning object-oriented programming? SIGCSE Bullet. 2005, 38, 39–43.

[11] Mendes, A.; Mendes, T. VIP—A tool to visualize programming examples. In Proceedings of the EACT 88—Education and Application of Computer Technology, Malta, October 1988.

[12] Gomes, A.; Mendes, A.J. SICAS: Interactive system for algorithm development and simulation. In Computers and Education: Towards an Interconnected Society; Ortega, M., Bravo, J., Eds; Kluwer Academic Publishers: Dordrecht, The Netherlands, 2001; pp. 159-166.

[13] Buck, D. e Stucki, D. J. (2001). JKarelRobot: a case study in supporting levels of cognitive development in the computer science curriculum”. ACM SIGCSE Bulletin, 33 (1), 16-20.

[14] Gomes, A. Ambiente de suporte à aprendizagem de conceitos básicos de programação. MSc Thesis, Faculdade de Ciências e Tecnologia da Universidade de Coimbra: Coimbra, Portugal, November 2000.

[15] Cooper, S., Dann, W. e Pausch, R. (2000). Alice: a 3-D tool for introductory programming concepts. Journal of Computing in Small Colleges, 15 (5), 107-116.

[16] Esteves, M.; Mendes, A.J. OOP-Anim, a system to support learning of basic object oriented programming concepts. In Proceedings of the CompSysTech'2003—International Conference on Computer Systems and Technologies, Sofia, Bulgaria, 2003.

[17] Rebelo, B.; Mendes, A.; Marcelino, M.; Redondo, M. Sistema Colaborativo de Suporte à Aprendizagem em Grupo da Programação—SICAS-COL. In Proceedings of the VII Simpósio Internacional de Informática Educativa, Leiria, Portugal, November 2005.

[18] Redondo, M.A.; Bravo, C.; Ortega, M.; Verdejo, M.F. PlanEdit: An adaptive tool for design learning by problem solving. In Proceedings of the 2nd International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems (AH2002), Malaga, Spain, May 29-31, 2002; Springer: Berlin; pp. 560–563.

[19] Naps, T. (2005). Jhavé – Supporting Algorithm Visualization. IEEE Computer Graphics and Applications, 25 (5), 49-55.

[20] Areias, C.M.; Mendes, A. ProGuide: A dialogue-based tool to support initial programming learning. In Proceedings of the 3rd E-Learning Conference—Computer Science Education, Coimbra, Portugal, September 2006.

[21] Marcelino M.; Mihaylov T.; Mendes A. H-SICAS, Handheld algorithm animation and simulation tool to support initial programming learning. In Proceedings of the 38th ASEE/IEEE Frontiers in Education Conference, New York, NY, USA, October 22-25, 2008.

[22] Dehnadi, S., Testing Programming Aptitude, In 18th Workshop of the Psychology of Programming Interest Group, page 22-37. University of Sussex, (September 2006).