F. I. Anfurrutia, A. Álvarez, M. Larrañaga and J. -M. López-Gil, "*Integrating Formative Feedback in Introductory Programming Modules*," in I**EEE Revista Iberoamericana de Tecnologias del Aprendizaje**, vol. 13, no. 1, pp. 3-10, Feb. 2018, doi: 10.1109/RITA.2018.2801898

Integrating Formative Feedback in Introductory Programming Modules

Felipe I. Anfurrutia, *IEEE Member*, Ainhoa Álvarez, Mikel Larrañaga, *IEEE Member,* Juan-Miguel López-Gil

***Title*— Integrating Formative Feedback in Introductory Programming Modules.**

***Abstract*— Introductory programming modules are challenging for both lecturers and students. In previous works, the authors have carried out educational innovations to mitigate these challenges and facilitate learning. This paper presents a further step in this improvement, proposing a learning process enriched with formative feedback. To this end, visual programming environments and educational robots are combined and complemented with automatic source code verification and validation feedback. The feedback integration proposal is presented along with the lessons learned from the previous experiments carried out that establish the basis of this work. The proposal has been implemented and tested in the Object Oriented Programming module in the Bachelor in Computer Management and Information Systems Engineering at the Faculty of Engineering of Vitoria-Gasteiz at the UPV/EHU University. The results of the evaluation have been positive and are also presented here.**

***Index Terms*—Computer science education, Programming environments, Educational robots**

I. INTRODUCTION

PROGRAMMING modules present many challenges for both teachers and students. On the one hand, these modules pose great difficulty for students and have low success rates, what involves a large number of retakers in these modules. In addition, some primary and secondary schools nowadays have begun to work on computational reasoning with various applications and robotic kits [1]. For this reason, part of novice students enters university studies with prior knowledge in certain aspects of programming. This produces a high degree of heterogeneity among students regarding their previous level in the competences addressed in those modules what is one of the main problems in teaching these modules [2]. This fact makes it difficult for teachers to design adequate learning methods for all students [3]. Additionally, programming modules are usually taught using general purpose programming languages, which may be very complex for novice students without prior knowledge of the module topics [2], [4]. Some programming languages have high learning curves. Others, even for the simplest programs, require a lot of code that is complicated to understand and address for novice students. In general, students have to cope at the same time with the design of the algorithms and the syntactic rules of used programming languages.

Literature presents two main ways to mitigate the problems that novice students find in introductory programming modules [5]. On the one hand, visual environments have been used to learn programming. This type of environments isolates students from the complexities of the underlying programming languages, allowing students to focus on the understanding of fundamental concepts before beginning to program [5]. On the other hand, the use of physical devices have also been suggested as an approach that allows students to interact with their programs in the real world [6].

The authors of this work have explored several proposals for improving teaching in order to mitigate the problems that students have in these modules. During the last five years, they have applied a pedagogical framework, based on Kolb's learning cycle [7] that integrates the use of educational robots and visual programming environments.

Although visual programming tools used so far have proved theworthier value for learning, they present limitations in the provided feedback, as this is mainly focused on the made mistakes. This feedback can help students correcting the errors identified in their programs, but does not provide a guide in the search for a solution. Therefore, providing assistanceto f guide the students in the search for a solution relies on the teaching staff. Given the importance of feedback in learning processes for the acquisition of new knowledge and skills [8], this article presents a proposal for the integration of enriched formative feedback. This proposal will integrate new types of feedback in order to provide a more enriching experience for students. This type of feedback or formative evaluation can contribute significantly in the students' learning process [9].

The rest of the work is structured as follows. First, aspects

F. I. Anfurrutia, Universidad del País Vasco/Euskal Herriko Unibertsitatea UPV/EHU, felipe.anfurrutia@ehu.eus

A. Álvarez, Universidad del País Vasco/Euskal Herriko Unibertsitatea UPV/EHU, ainhoa.alvarez@ehu.eus

M. Larrañaga, Universidad del País Vasco/Euskal Herriko Unibertsitatea UPV/EHU, mikel.larranaga@ehu.eus

J-M. López-Gil, Universidad del País Vasco/Euskal Herriko Unibertsitatea UPV/EHU, juanmiguel.lopez@ehu.eus

DOI (Digital Object Identifier) Pendiente

related to formative feedback are introduced. Section III summarizes previous experiences in introductory programming modules implemented by the authors during the last five years, which are explained in more detail in [10], together with the lessons learned from them. These lessons learned are the starting point for the formative feedback integration proposal in introductory programming modules presented in this work. Next, section V describes the experience in which the presented framework has been used. Finally, conclusions of the work are detailed.

II. Formative feedback

The formative feedback includes any information that is communicated to a student with the aim of modifying the student's thinking or behavior and improving the learning [8], [11]. On the one hand, students can rely on provided feedback to take corrective measures and improve their learning. On the other hand, teachers can rely on feedback to take measures to improve their teaching.

In the case of programming, students often face a trial-error process, which could be significantly improved with adequate feedback that would allow students to reflect on their learning process [12].

In general, provided feedback can fulfill two functions: directive or facilitative [13]. Directive feedback indicates students what should be reviewed or modified; whereas facilitative feedback provides comments or suggestions to guide students in their own review and conceptualization.

Effective feedback should provide students two types of information [8]: verification and elaboration Verification information indicates whether a program is correct or not, while elaboration information includes aspects that can guide students, providing clues that guide them towards the correct answer.

Verification information can be provided explicitly or implicitly. Information provided explicitly is expressed by symbols indicating the correctness or incorrectness of a solution. Implicitly provided information includes aspects such as simulations that demonstrate the outcome of the solution proposed by a student. Feedback can also be generated, including verification aspects enriched with prepared explanations. These explanations can address the errors, the concept that was being tackled or could be developed examples.

This article proposes improving the learning framework based on the Kolb cycle [7], used by the authors in the introductory programming modules, through the integration of feedback. On the one hand, the proposal incorporates implicit verification information through simulations and executions in physical environments. On the other hand, it includes prepared feedback that allows guiding students towards the solution of a problem. Many of the errors that students make in their programs are similar and their detection and solving can be automated to a great extent. Therefore, the proposal incorporates an automated process of error detection and associated feedback based on the use of unit tests. The following describes the previous experiences and lessons learned from them, which have been fundamental to the design of the proposal presented in this article.

III. Previous Experiences in Programming

This section introduces the framework proposed in [10] for implementing the Kolb's learning cycle in programming, an approach that incorporates the combined use of educational robots and visual environments

This framework has been used during 5 academic years in the two introductory programming modules taught in the first year of the Bachelor in Computer Management and Information Systems Engineering at the Faculty of Engineering of Vitoria-Gasteiz at the UPV/EHU University: *Introduction to Programming* during the first semester, and *Modular & Object Oriented Programming* (MOOP) in the second [10], [14].

A. Kolb's cycle in Programming

Simultaneous learning of the conceptual aspects of programming and the syntactic rules of programming languages is an added difficulty for novicestudents [2]. In this sense, some authors have suggested to deal with the conceptual aspects of programming in the first place to later work on the notational aspects associated with a specific programming language [15]. In addition, in the case of programming, several authors have pointed out that practical situations, in which students actively build knowledge, are the most appropriate [16], [17].

Taking these recommendations into account, the authors decided to tackle the contents from two different perspectives: first from a conceptual perspective and, later, from a notational one. The goal of the conceptual perspective is that the students understand and be able to apply the contents addressed in problem solving, while the notational perspective considers the syntactic and semantic aspects of a specific programming language or formalism such as flow charts. The authors determined to apply Kolb's learning cycle [7] in each of the perspectives, which has proven to be very useful for learning programming [18]. Kolb proposes that for learning to be effective, students should perform a cycle of the four stages shown in Fig. 1. First, students must perform a specific activity. Afterwards, they must reflect on the experience, in order to be able to conceptualize the theory that allows explaining performed observations. Finally, they must apply the theory in new situations.

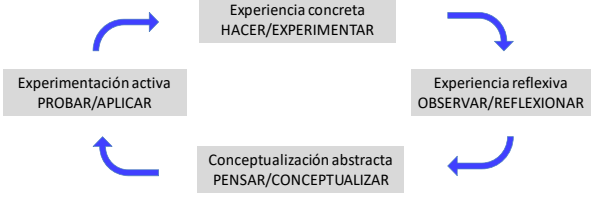

Fig. 1 Kolb's learning cycle

Examples of activities carried out in the implementation of the cycle are presented next. These activities are described in more detail in [19].

1) Concrete experience: The main objective of this stage is to actively involve students in specific activities related to the topics addressed. For example, to work on the concepts of classes, attributes and methods, students were given a project in which they could create instances of geometric figures and interact with objects generated by invoking their methods.

2) Reflective experience: Once a concrete experience has been performed it is necessary for students to reflect on this experience. With this aim, the activities designed for the concrete experience stage have been complemented with a series of questions that induce the students to reflect on the concepts addressed. For the example in the previous step, students had to fill in a table reflecting both the message passing and the status of the objects after executing the methods. Likewise, the teachers encouraged group discussion on what was observed in the previous stage.

3) Abstract conceptualization: At this stage, the contents addressed are treated in a formal way under the supervision of the teaching staff. To this end, the teaching staff provided examples related to the topics covered in the experiences carried out in the previous two stages. These examples allowed the students to assimilate the topics covered.

4) Active experimentation: In the last stage, students should put into practice and consolidate the topics discussed in the previous stages. To this end, they were asked to make drawings by combining the geometric figures of the previous steps.

This learning cycle has been applied for each of the topics in the selected modules for the experiences carried out so far by the authors [10].

B. Supporting Tools Used

In the experiments carried out so far, educational robots and visual programming environments have been used. In order to properly select the tools, a study of the existing tools was carried out to determine the ones that best fit the characteristics of each module, facilitate the learning of the students and minimize the problems identified in the bibliography.

These tools were selected considering both published comparisons [4], [20] and the support they provided for conceptual and notational perspectives. It was also considered the programming language to which they were oriented, since in the definition of the Bachelor it was determined that the programming language to be used in the different programming modules would be Java.

In the case of educational robots, it was decided to use Lego Mindstorms robots. The main reasons for this selection were that they allowed the use of different programming environments and the familiarity that many students have with Lego products, with which many of them have played in their childhood.

In the case of visual programming environments, different tools were selected in each module. In *Introduction to Programming*, the objective was to learn the main aspects of the design and implementation of simple programs, so Scratch (*https://scratch.mit.edu/*) was chosen. This tool was designed to inculcate programming notions to a younger audience, but has also been successfully used in higher educational institutions [21]. On the other hand, in *Modular and Object Oriented Programming* module, the objective was to work with the main Object Orientation concepts. In this case, BlueJ was chosen for the first half of the module and Greenfoot for the second half. These two tools have proved their worth for teaching in this area [20]. In addition, they support the notational perspective, since they have been developed for the learning of Object Oriented Programming in Java.

C. Lessons Learned in Previous Experiences

Previously carried out experiences have allowed identifying a set of aspects to consider when introducing similar experiences in programming modules:

1) Characteristics of the students: Although obtained results have been generally positive, it has been detected that there are differences in the results, attending to gender and students' previous programming knowledge.

Differences in the results according to the gender of the students present the need to continue analyzing the results to check, for instance, whether the problem is in selected development environments or in the subjects of carried out exercises. Extending this part of the study would allow to adequately adapt the positive results obtained in the presented experiences.

Another latent problem that must be properly dealt with is the previous knowledge difference that students have. In the responses to the surveys, there have been marked differences regarding the motivation and the acceptance of used tools attending to the previous knowledge of the students.

One possibility to address this problem may be the use of different programming environments for different categories of students. In addition, the programming environment can be changed as the students' progress, since there is no adequate environment for every situation [4].

2) Support Tool Selection: Certain difficulties inherent to the use of physical devices have been detected. However, they should not be automatically discarded because a greater motivation has been observed with this than with non-physical environments. This raises the need for exercises to be designed with such contextual factors in mind [22].

In addition, visual environments allow easily abstracting from specific details and focusing on the implementation of the application logic. Moreover, since there is no environment good for every situation, combinations of different tools should be analyzed.

3) Integration in the Module: To achieve greater acceptance by students, it is important to properly integrate the environments into the module development. Therefore,

teachers should consider the activities carried out using these environments in the assessment process. In addition, teachers should relate selected environments to other modules of the programming branch, as well as to the professional activity, so that students have a global perspective and appreciate the utility of used tools.

4) Methodological Aspects: Despite not having achieved a statistically significant improvement in qualifications, the positive effect of the established pedagogical framework is obvious in aspects such as the increase in the attendance rate. In order to work with a pedagogical framework such as the one presented in this article and to correctly apply Kolb's cycle, it is fundamental to ask questions after doing the exercises, so that students can observe, reflect and find answers to the problems raised. Visual development environments facilitate the accomplishment of these tasks and give adequate support to methodological aspects.

IV. Formative Feedback in Introductory Programming Modules

In the lessons learned, positive and negative aspects have been identified both with educational robots and with visual educational environments. Physical robots have had a greater positive influence on motivation, but there were problems related to the physical environment in which they were used and limitations for their use outside of the classroom. Visual environments have not had such a positive influence on student motivation, but they do not suffer from the other problems detected in the experience with physical robots. With these aspects in mind, it seems that the integration of both types of tools can be useful in introductory programming modules. In order to confirm this hypothesis, the authors performed a pilot study proposing a scenario based on virtual worlds that had a good acceptance by the students [14]. Taking these preliminary results and the lessons learned from previous experiences into account, this paper presents an integrated solution that combines the advantages of visual environments and educational robots enriched with formative feedback and explicit verification. To this end, validations have also been incorporated by unit tests.

Combining automatic validation with simulations and executions in real environments of educational robots allows providing a rich and different feedback to students. In addition, the use of virtual environments that show simulations alleviates the need to use physical robots, allowing the students to perform tasks in this environment outside the school hours in more motivating scenarios.

The feedback integration model (see Fig. 2) involves 5 phases, which are described next.

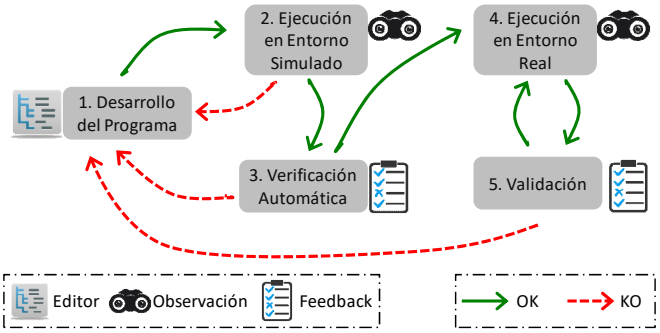

Fig. 2 Feedback integration model

A. Development

It is envisaged that students use a development environment (e.g., programming environments by blocks of instructions). This environment should support the previously identified two cycles (conceptual and notational).

The tools or combinations of tools used in this phase must support both block-based editing and direct editing using a particular programming language. This would allow those students with prior knowledge to tackle only the notational perspective.

Again, as it has been done in previous experiences, the tools will be used first from a conceptual perspective and then from a notational one.

For the application to be satisfactory, it is vital to adequately define the types of exercises proposed to students. In this aspect, generating a list of exercises adaptable to the subject preferences of students is proposed. Thiswill allow responding, among other aspects, to the differences detected on a gender basis.

B. Execution in Simulated Environment

The integrated environment will show on the screen a simulation of the behavior of robots running the program developed in the first phase. The tool used should allow for pauses, as well as for observing and analyzing the state of the stage and of the robots within it.

Simulation provides feedback with implicit verification information that allows early identification of errors and incorrect solutions, allowing for more efficient use of time. In addition, it makes the observation and reflection of what has happened easier, supporting the corresponding phases of Kolb's learning cycle.

C. Automatic Verification

This phase is mainly focused on the verification of the correction of the developed program by means of unit tests. These unit tests make it possible to bound errors (e.g. by class or method) and to carry out a more in-depth study of the program, in which situations that students have not taken into account can also be detected. In the previous phase students already obtained verification information, whereas in this phase students will obtain feedback with elaboration infomation. That is, the result of the verification will not be limited to a simple "accepted/rejected" message, but it will be

enriched with explanations about possible errors or malfunction causes. This will guide the students towards solving the problem.

Both functional checks are carried out in this phase, those which focus on the correctness of the programs, and those focusing on the conceptual ones (it is verified that aspects seen in the module are correctly applied).

For example, in the case of MOOP, feedback related to conceptual verifications could indicate whether the declaration of classes or the initializations of attributes are correct in the constructor. Examples of feedback offered in these cases could be: "*class X should inherit from Y. Possible cause: the correct keyword is not used.*", and "*attribute A has not been initialized in the constructor of class C. Possible cause: definition of a local variable with the same name as the attribute*".

Regarding functional verifications, they might suggest that the robot should go backwards when it detects that there is an obstacle in front of it. The feedback provided in this case could be: "*The robot does not go back in front of an obstacle. Possible Cause: The program does not check that the contact sensor has been pressed or has not moved far enough back.* "

When errors are detected in the automatic verification phase, students will receive a report that will allow them not only to correct the mistakes made, but also to help them interpret the events that have occurred and to contrast what has been done with what has been learned. The reports generated in this phase will provide information for the Kolb's cycle conceptualization stage.

Providing feedback as a result of verification will allow carrying out a formative assessment [12] and facilitates the work of the teachers. Likewise, obtaining a score in the validation phase will allow a greater integration of the environment in the evaluation of the module.

D. Execution in Real Environment

In case the automatic verification is satisfactory, the execution will be performed in a real environment by bringing the program code in the robot and executing it in a real physical environment. This phase, similar to what happens in the execution in a simulated environment, will allow students to observe and analyze the behavior of robots, in a real environment. This will, again, support the observation and reflection phases. In this case, this execution will allow students to analyze problems that occur in real environments, such as lighting conditions or surface friction.

The possibility of testing developed programs in a physical robot can also be an aspect that increases student motivation.

E. Validation

Once the previous aspects are completed, in this last phase the validation of both functional compliance and conceptual or design aspects of the program is performed.

For the functional compliance aspect, user acceptance tests will be performed, which consist of validating that the program executed in the robot satisfies client's requirements [23]. During this phase, the interested audience executes the program in a robot in real scenarios. This phase allows detecting those errors that could not be detected in the automatic verification. Provided formative feedback will be complementary to the one received in the automatic verification: e.g. *the radius of the rotated is too large, the configuration of the ports is not adequate or the location of the sensor should be adjusted*.

For the conceptual compliance aspects, the teaching staff will review, in the delivered program, the aspects of design or principles worked on the module (such as modularization, reuse and programming styles), in order to enrich the provided feedback even more.

V. Proposal Implementation and Evaluation

This section describes the tools used together with methodological aspects of the implementation of the proposal for the incorporation of formative feedback in introductory programming modules presented in the previous sections. The results of this experience are also described. In this case, 42 students of the MOOP module at the Faculty of Engineering of Vitoria-Gasteiz at the UPV/EHU University participated in the study.

The students of this module show heterogeneity regarding previous knowledge about Object Oriented Programming. A percentage of the students already access the Bachelor degree with some knowledge about this programming paradigm, while another large percentage lack knowledge of the fundamentals of programming. The previous programming modules do not cover the MOOP paradigm, so they cannot perform a homogenizing work regarding students' previous knowledge on this topic.

A. Tools Used

One of the main aspects related to the implementation of the proposal is a suitable tool selection to adequately develop all phases of the proposal.

From the results obtained in previous experiences, it was determined to continue working with Lego Mindstorms robots, as they had previously obtained very positive results. On the other hand, given the results of a previous pilot test [14], it was determined to look for a simulation environment that would allow Lego Mindstorms robots to be simulated in a way closer to the physical robot.

After analyzing different options available, it was decided to use the RobotSim simulation package (http://www.aplu.ch/home/apluhomex.jsp?site=75). This tool allows observing the simulated executions of the programs that will be executed in the physical Lego robots. It also provides a library of Java classes that allow manipulating the different components of Lego robots as well as defining different scenarios. This library can be used in various development environments such as Eclipse or BlueJ. In this way, students are allowed to select the tool to use based on their previous knowledge and preferences, thus responding to the need to adapt the proposal to the characteristics of the

students.

On the other hand, to perform the automatic verification stage, Web-CAT was chosen. This tool can be easily integrated with the development environments selected for the previous stages and provide elaborated feedback. This type of feedback was implemented by defining unit tests using the JUnit library.

Fig. 3 summarizes the model of feedback integration together with the tools that have been used for the experience presented in this article.

B. *Methodological Aspects*

Next, the way in which the proposed model has been applied is described. The teaching staff provided students with a simple program as an example of using the different components of the Lego robot, to later be used in the project they had to develop. Following a pre-defined script, the students made small modifications in the program in the development phase. In the simulation phase, students observed the effects of the modifications made. These observations, which were complemented by the feedback provided by the validation, can be useful for the conceptualization phase of the Kolb's learning cycle.

Then, students transferred their program to the robot and executed it in the physical environment, while observing how the robot behaved in the physical environment. Later, students experienced the functionality offered by the Robotsim library by developing new programs and taking advantage of the execution in both the simulated environment and the real environment to observe and understand the behavior of the robots and their components.

C. *Data Gathering and Analysis of Results*

The main objective of this study was to analyze student satisfaction with the implemented proposal. For this, a survey was conducted before the final exam and the publishing of module's grades.

The survey included questions about data contextualization and a set of four-point Likert items with categories "totally disagree", "disagree", "agree" and "totally agree", mainly related to the feedback provided by Web-CAT and the simulations (see Table I).

A total of 42 students completed this survey, the results of which are detailed next. 17% of the members of this group were women and 24% were retakers.

Regarding the feedback provided by Web-CAT, 85% of the students indicated that it motivated them to finish the proposed exercises (question Q1). In addition, 80% of students indicated that this feedback helped them identifying and correcting the errors of their programs (Q2).

TABLE I: EXTRACT SURVEY

	Question
Q1	Obtaining feedback with Web-CAT motivates me to finish proposed exercises
Q2	The feedback provided by Web-CAT has helped me to correct the errors in my program
Q3	The simulation has helped me to understand the effect of each instruction
Q4	The simulation has helped me to detect errors in my program
Q5	The simulation has helped me to correct errors in my program
Q6	The simulation has increased my motivation in performing exercises

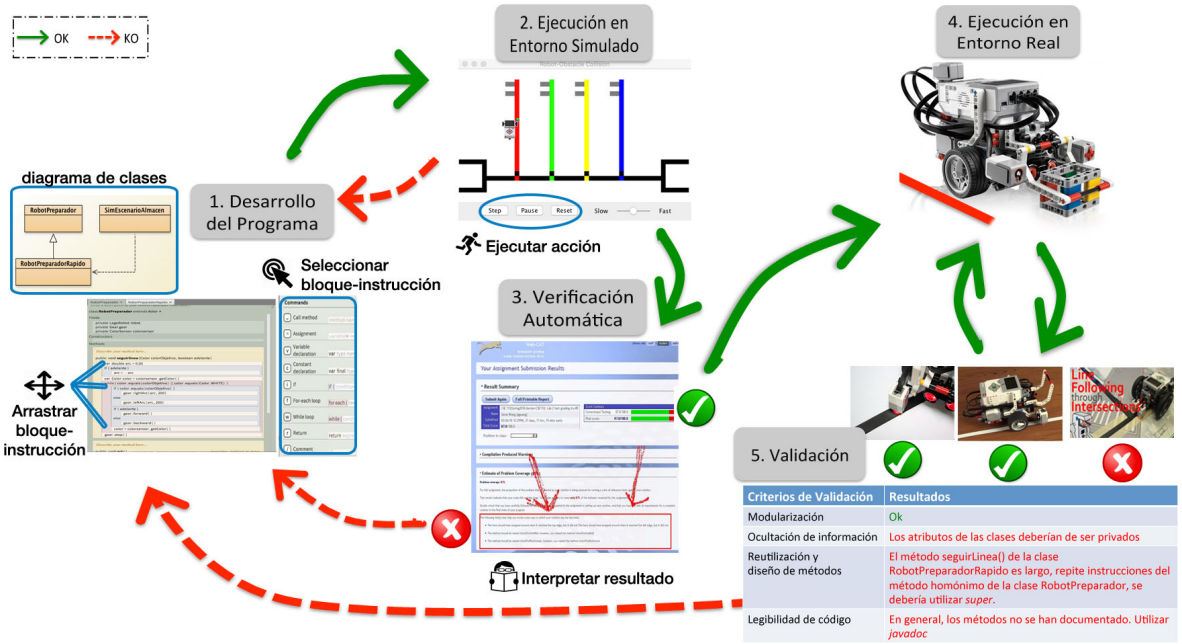

Fig. 3 Tools used for implementing the proposal

In relation to the simulations, 88% of the students indicated that simulations have helped them understanding the effect of each programmed instruction (Q3). In addition, 90% of students indicated that using the simulations has helped them detecting errors in their programs (Q4) and 88% think that it has also helped them correcting them (Q5). Finally, 88% of students indicate that their motivation has increased with the use of simulations (Q6).

The results of the previous experiences carried out by the authors showed notable differences attending to the gender, obtaining more negative responses by the women [19]. This study has also considered the gender perspective, given that the small number of women who choose to study computer science is one of the main concerns in the community [24], [25]. The results obtained in this experience have been more positive than those collected in the previous experiences. It should also be noted that in this experience the answers to the questions related to the simulations by the women have been more positive than those of the men. In this case, 100% of women have indicated that their motivation has increased. The same applies to error detection and correction, for which 100% of women indicated that the simulations helped them.

The results have also been analyzed considering whether the students were retakers or not. In this case, the results have been very similar between retakers and non-retakers, except in the aspects related to feedback. Only 60% of the retakers indicated that the feedback provided by Web-CAT (P2) helped them identifying errors in their programs compared to 87% in the case of non-retakers. Regarding the feedback provided by the simulations, 80% of the retakers indicated that it helped them detecting errors, while the percentage rises to 94% among the non-repeaters. These differences may be due to the retaker students having more experience compared to non-retakers, which makes it easier to detect and correct habitual errors.

Although responses have been generally positive, those related to the simulations are slightly higher than those related to Web-CAT. This can be motivated by the type of scenarios used, which made it difficult for all developed unit tests to provide effective or adequate feedback for students.

In addition, the results have been analyzed to check whether the innovation presented in this work had a positive effect on learning or not. Kolling stated that using different methodologies with two groups of students when one produces better results than the other is not appropriate [26]. Therefore, in this case, a *between-subjects* study has been conducted in which students are not divided into two different groups, but rather the students' results are compared across different academic years. The results of the last academic year in which the new methodology was applied has been compared to the results of the two academic years prior to this study. The results of the last year have greatly improved in all aspects the results of previous years. On the one hand, the percentage of student passing this last academic year has increased considerably, standing at 58.82% compared to a range between 21% and 48% in previous ones. On the other hand, the number of first-time attendees has risen to a 65.88%, compared to the 38% and 62% in previous academic years. The success rate of the module, measured by means of the percentage of students that pass the module in relation to presented ones, has been increased from a percentages between 35% and 70% up to a 80%. The yield rate, defined as the percentage of passing students in relation to enrolled ones, has reached the 53%, compared to values in the range of 14% to 47%. These data show that results achieved in this module have been the best since the module began to be taught in the new curriculum 7 years ago.

VI. Conclusions and Future Work

Adequate formative feedback is vital for students to improve their learning. This becomes even more relevant in subjects such as those related to introductory programming that present different teaching problems.

In this paper, a proposal for incorporating formative feedback in introductory programming modules has been presented. To this end, Kolb's learning cycle (successfully used in previous experiences), simulation environments, physical robots and unit tests have been combined.

The analysis of the results of previous experiences suggested that an adequate combination of the use of visual environments together with physical robots could improve students' motivation in MOOP [14]. However, the high degree of heterogeneity in the prior knowledge of first-year students affects the application of improvements. Therefore, the use of this type of tools requires adjusting the type of learning environments used to the previous knowledge of the students, as well as adequately managing the amount of environments to be used in a subject or semester, especially if they are new to students.

A first evaluation of the proposal has been made using BlueJ, the Eclipse development environment, the RobotSim package for simulations, WEB-CAT for unit testing and Lego Mindstorms robot for executions on physical environment.

To evaluate performed implementation, the responses to a survey and the academic results of the students were analyzed. The results of the evaluation have been very positive, so the implementation of the proposal is promising. In addition, it should be noted that the responses given by women have been more positive than those of men. This factor is important to keep the interest of women entering computer science bachelors [24].

Given the positive results of the experience, we are currently implementing it in the *Introduction to Programming* module. In addition, it will continue to be used in *MOOP* in order to analyze the effects on motivation and satisfaction, as well as in the academic results. The integration of new tools to enrich the feedback provided in the activities [27] will also

continue to be analyzed, with the aim of making it easier to be implemented and evaluated by the teaching staff [9].

ACKNOWLEDGEMENTS

This work has been partially funded by the Basque Country Government (IT980-16) and the University of the Basque Country (GIU16/15 and EHUA16/22).

REFERENCES

[1] F. J. García-Peñalvo, A. M. Rees, J. Hughes, I. Jormanainen, T. Toivonen, and J. Vermeersch, «A survey of resources for introducing coding into schools», 2016, pp. 19-26.

[2] A. Gomes and A. J. Mendes, «Learning to program-difficulties and solutions», en *International Conference on Engineering Education–ICEE*, Coimbra, Portugal, 2007, vol. 2007.

[3] D. C. Leonard, *Learning theories, A to Z*. Westport, Conn.: Oryx Press, 2002.

[4] A. J. Hirst, J. Johnson, M. Petre, B. A. Price, and M. Richards, «What is the best programming environment/language for teaching robotics using Lego Mindstorms?», *Artif. Life Robot.*, vol. 7, n.º 3, pp. 124-131, sep. 2003.

[5] A. Wilson and D. C. Moffat, «Evaluating Scratch to introduce younger schoolchildren to programming», *Proc. 22nd Annu. Psychol. Program. Interest Group Univ. Carlos III Madr. Leganés Spain*, 2010.

[6] D. O'Sullivan and T. Igoe, *Physical Computing: Sensing and Controlling the Physical World with Computers*. Boston: Thomson, 2004.

[7] D. A. Kolb, *Experiential learning: experience as the source of learning and development*. Prentice-Hall, 1984.

[8] V. J. Shute, «Focus on Formative Feedback», *Rev. Educ. Res.*, vol. 78, n.º 1, pp. 153-189, mar. 2008.

[9] R. Rashkovits and I. Lavy, «FACT: A Formative Assessment Criteria Tool for the Assessment of Students' Programming Tasks», *Lect. Notes Eng. Comput. Sci.*, vol. 2204, n.º 1, pp. 384-389, jul. 2013.

[10] F. I. Anfurrutia, A. Álvarez, M. Larrañaga, and J. M. López-Gil, «Incorporating educational robots and visual programming environments in introductory programming modules», en *Int. Symposium on Computers in Education (SIIE)*, 2016, pp. 1-4.

[11] M. Taras, «Assessment – Summative And Formative – Some Theoretical Reflections», *Br. J. Educ. Stud.*, vol. 53, n.º 4, pp. 466-478, dic. 2005.

[12] S. H. Edwards, «Using software testing to move students from trial-and-error to reflection-in-action», *ACM SIGCSE Bull.*, vol. 36, n.º 1, pp. 26–30, 2004.

[13] P. Black and D. Wiliam, «Assessment and Classroom Learning», *Assess. Educ. Princ. Policy Pract.*, vol. 5, n.º 1, pp. 7-74, mar. 1998.

[14] F. I. Anfurrutia, A. Álvarez, M. Larrañaga, and J.-M. López-Gil, «Lecciones aprendidas de experiencias con robots educativos and entornos de programación visuales en asignaturas de programación», *IE Comun.*, vol 25, pp. 9-22, 2017.

[15] H. Zhu and M. Zhou, «Methodology First and Language Second: A Way to Teach Object-oriented Programming», en *Companion of the 18th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications*, New York, NY, USA, 2003, pp. 140–147.

[16] E. Lahtinen, K. Ala-Mutka, and H.-M. Järvinen, «A study of the difficulties of novice programmers», *ACM SIGCSE Bull.*, vol. 37, n.º 3, pp. 14–18, 2005.

[17] C. Wang, L. Dong, C. Li, W. Zhang, and J. He, «The Reform of Programming Teaching Based on Constructivism», en *Advances in Electric and Electronics*, W. Hu, Ed. Springer Berlin Heidelberg, 2012, pp. 425-431.

[18] L. Yan, «Teaching Object-Oriented Programming with Games», en *Sixth International Conference on Information Technology: New Generations, 2009. ITNG '09*, 2009, pp. 969-974.

[19] F. I. Anfurrutia, A. Álvarez, M. Larrañaga, and J.-M. López-Gil, «Entornos de Programación Visual para Programación Orientada a Objetos: Aceptación and Efectos en la Motivación de los Estudiantes», *Rev. Iberoam. Tecnol. Aprendiz.*, vol. 5, pp. 11-18, 2016.

[20] S. Georgantaki and S. Retalis, «Using educational tools for teaching object oriented design and programming», *J. Inf. Technol. Impact*, vol. 7, n.º 2, pp. 111–130, 2007.

[21] J. Maloney, M. Resnick, N. Rusk, B. Silverman, and E. Eastmond, «The Scratch Programming Language and Environment», *ACM Trans. Comput. Educ.*, vol. 10, n.º 4, pp. 1-15, nov. 2010.

[22] A. Álvarez and M. Larrañaga, «Experiences Incorporating Lego Mindstorms Robots in the Basic Programming Syllabus: Lessons Learned», *J. Intell. Robot. Syst.*, vol. 81, n.º 1, pp. 117-129. 2016.

[23] R. Rice, «What is User Acceptance Testing?» [En línea]. Disponible en: http://www.riceconsulting.com/articles/what-is-UAT.htm. [Accedido: 24-abr-2017].

[24] M. A. Rubio, R. Romero-Zaliz, C. Mañoso, and A. P. de Madrid, «Closing the gender gap in an introductory programming module», *Comput. Educ.*, vol. 82, pp. 409-420, mar. 2015.

[25] J. Robertson, «The influence of a game-making project on male and female learners' attitudes to computing», *Comput. Sci. Educ.*, vol. 23, n.º 1, pp. 58-83, mar. 2013.

[26] M. Kölling, «Using BlueJ to introduce programming», en *Reflections on the Teaching of Programming*, Springer, 2008, pp. 98–115.

[27] O. Shaikh, «Real-time feedback for students using continuous integration tools · GitHub». [En línea]. Disponible en: https://github.com/blog/2324-real-time-feedback-for-students-using-continuous-integration-tools. [Accedido: 28-jun-2017].