



**Relation between Programming Visual Learning with VILEP
and Students' Emotions**

Journal:	<i>IEEE Revista Iberoamericana de Tecnologías del Aprendizaje</i>
Manuscript ID	Draft
Manuscript Type:	Original Article
Index Terms:	Computer science education, Learning, Emotion recognition

SCHOLARONE™
Manuscripts

Relation between Programming Visual Learning with VILEP and Students' Emotions

Darwin Alulema and Maximiliano Paredes

Abstract—In this article a visual programming tool is shown, which allows coding programs by abstracting complex parts of the syntactic structures of the language by means of graphic representations combined with textual expressions. The tool allows student to relate the syntax of a language to the logic of solving a problem. To validate the proposal, we carried out an experience with a group of students who used the tool and another group who used Eclipse. The results show that there was an additional 23.4% of students who used the tool and reduced their error rate compared to those who used Eclipse.

Index Terms— Computer science education, Learning, Emotion recognition.

I. INTRODUCTION

THIS paper is an extension of the article published in the proceedings of the International Congress on Learning, Innovation and Cooperation CINAIC 2019 entitled "Evaluation of Students' Emotions in Visual Programming Learning." The article has been extended in the present work in the following way: a) a new section of related works has been incorporated (section II), b) the description of the tool design has been extended (section III) and c) the analysis of the results obtained has been extended (subsection E of section V).

The rapid advance of technology has changed the way teachers deliver instruction and students learn. Although these advances are also present in learning programming [1], learning programming is not an easy task. Learning to program involves a number of difficult achievements, both for students in the computer science grades, and for students in other related grades. However, the former, due to their own field of study, will have to develop this capacity further [2]. Aktunc (2013) lists numerous challenges that instructors face and that complicate the learning of programming in the introductory subjects of computer science and related degrees: a) great variation in the knowledge profile of students; b) discouragement and demotivation of most students as they perceive programming as a difficult and complex cognitive task; c) excessive time spent teaching the syntax of the

programming language (it should be noted that spending too much time learning language syntax without applying it in a context of use is detrimental to students [3]); and d) most programming environments used for teaching are confusing, as they were built for professional software development and do not have a didactic approach, so all these factors generate problems in introductory programming courses [4].

The traditional way of introducing programming for beginners is through a language-oriented introductory course [5]. But this language-oriented approach results in several problems: a) students have difficulties because of the complexity of the syntax, b) the syntax requires extra learning time and c) the language itself does not provide advantages in the understanding of the programming concepts that underlie its structures, even its syntax can make it difficult to understand the concepts. However, the use of a programming language is necessary for the learning and practice of programming concepts. This is particularly the reason why the syllabus of a first-year programming course will often spend a considerable amount of class time on learning the syntax of the language [6]. Therefore, language should not be neglected in the learning process and solutions should be sought that adequately manage its use as a learning tool in the educational context to mitigate the disadvantages identified. This research focuses specifically on finding solutions with this orientation. In order to motivate young students in the first year, the learning experience should be enriching by incorporating practical, creative and "fun" activities [7]. However, in introductory programming subjects, this is not how students perceive it [8]. In addition to the challenges of learning to form structured solutions for programming problems, students must also deal with the difficulty of the syntax and commands of the programming language they use to form solutions to problems, whose commands may have apparent confusing names. In this difficult context for the student, they often perceive that a personally significant learning context is not generated, experiencing a lack of motivation, and may even lose interest in learning [5].

The aim of this research is to propose educational resources for learning programming, for which the authors suggest progressively abstracting the syntax of the programming language (by means of scaffolding technique). Thus, the aim is to motivate the student from a positive emotional state. As a result, students will acquire a certain level of fluency in a programming language before starting to implement their solutions in source code directly. Some research proposes that,

D. Alulema, Universidad de las Fuerzas Armadas ESPE, Sangolquí, Ecuador; e-mail: doalulema@espe.edu.ec).

M. Paredes, Universidad Rey Juan Carlos, Móstoles, Madrid, España; e-mail: Maximiliano.paredes@urjc.es).

DOI (Digital Object Identifier) Pendiente

1 instead of implementing systems oriented to the complete
2 learning of a programming language, tools or learning objects
3 are implemented on a small scale for specific programming
4 concepts [6]. These tools should integrate scaffolding
5 techniques [9] that adapt the learning structure and contents to
6 the student. Research in education and cognitive psychology
7 suggests that many students today have a visual and
8 interactive learning profile. Therefore, it seems reasonable to
9 reduce the textual detail of the programming language syntax
10 that students have to learn in the early stages, and to develop
11 more visual content on the programming concepts themselves,
12 trying to facilitate the relationship of these concepts with the
13 problem-solving process.

14 For these reasons, this article proposes a tool called VILEP
15 (Visual LEarning Object-oriented Programming) that is
16 developed for the learning of OOP (Object-Oriented
17 Programming), which can hide or make visible details of the
18 language syntax used by the student by combining visual and
19 textual representations. Thus, it combines visual programming
20 with textual programming of source code in an appropriate
21 way, orienting the process to visual learning. Previous studies
22 show that the use of visual learning has a significant impact on
23 improving students' problem solving and analytical thinking
24 skills and promotes active learning [10]. In this regard there
25 are programming tools with a more or less visual approach to
26 introduce students to programming, such as Scratch [11],
27 Alice [12], Blockly [13], Greenfoot [14], among others.
28 However, visual tools over time cause students to become
29 dispersed and distract their attention, which is why proposals
30 such as Rahman's [5] point out not to abandon the use of
31 textual languages completely. Moreover, this type of tool does
32 not establish scaffolding techniques for learning programming
33 as in this proposal. For this reason and without losing sight of
34 the fact that the objective of an introductory programming
35 course should not only be to teach a programming language,
36 but also to teach the different forms of problem solving,
37 reasoning logic, basic algorithm design and general
38 programming concepts, trying to have little or minimal
39 emphasis on language syntax [5]. In this context, the use of
40 visual programming tools in the early stages of learning could
41 facilitate learning. Thus, the VILEP tool described in this
42 article has been developed through model driven engineering
43 and proposes a graphic editor that allows students to
44 implement programs in Java through visual resources by
45 hiding complex syntactic expressions of the language. This
46 article describes the tool from a teaching approach and
47 demonstrates in a preliminary way the validity of the tool in
48 the educational context. For this purpose, an experience with
49 first-year programming students in the classroom has been
50 carried out and the knowledge, perceptions, and emotions they
51 have experienced with the use of the tool have been measured.

52 The article is structured in the following way: section II
53 mentions some of the works that deal with the use of
54 specialized tools in teaching programming. Section III
55 describes how the VILEP tool was implemented using model
56 driven engineering. Section IV discusses the learning method that
57 has been used to address the use of the proposed tool. Section V
58 describes the experience made in the classroom and section VI
59 shows the conclusions and future work.

II. RELATED WORK

Programming is considered a fundamental skill since many
of the programming concepts are used in almost all the basic
courses at the university level [15]. Despite its importance,
teaching programming languages is difficult as it involves the
understanding of theoretical concepts, the practical use of
language semantics, syntactic coding and reasoning logic for
problem solving [16-17]. For this reason the learning of
programming has received greater attention. However, the
traditional approach to learning programming, which is based
on textual languages, is too difficult for many students to
learn, often resulting in low learning success rates in the early
years [17],[19].

On the other hand, it should be noted that the educational
model used in classrooms in engineering degrees is usually
[15]: a) auditory, b) abstract, c) deductive, d) passive and e)
sequential, in contrast to most students who are: a) visual, b)
sensitive, c) inductive, and d) active. This difference,
associated with the difficulty of learning a programming
language, mainly contributes to the lack of student interest in
computer courses, low student performance [18] and teacher
frustration [17]. Therefore, there is a problem of demotivation
and performance in the current models of learning
programming, which becomes more visible in the introductory
courses of that subject.

There are several factors that can influence achievement in
introductory programming courses [20]: a) the student's
previous programming experience gained from high schools;
b) experience with prior knowledge in other sciences where
the computer is used; c) the relationship between the students'
learning styles and the programming language learning model;
d) expectations of learning outcomes and student self-efficacy;
and e) the emotions experienced by the students [15]. These
factors can influence the learning process in a positive or
negative way, doing the contents become meaningful. In
addition, problems in learning outcomes could also be due in
part to an incorrect way of teaching programming, or the use
of inappropriate teaching materials [17], as these may not
guide the student in the learning process. All these elements
condition the learning model of programming, in which, and
according to traditional educational approaches, students have
to write the program directly using a programming language.
This can lead to students who have little experience in
programming feeling easily frustrated and also undermine
their motivation to learn [21]. In addition, it also takes a lot of
time and work for teachers to create textual programming
learning materials commonly used in traditional educational
models [22].

To partially solve these problems many tools and
programming environments have been developed to facilitate
efficient learning and understanding [22,23]. In this sense the
tools should present the most relevant programming concepts
taught in an introductory programming course, making sure to
include the concepts that students find difficult [17].
According to Moons [17] there are four main ways to teach
programming. The first is by using a certain order-oriented
programming education methodology to introduce various

programming topics. The second is through the use of various active learning techniques inspired by constructivism, such as role-playing, active story-telling, workshops, etc. The third approach is to use a programming language that is designed for students who are new to programming. The fourth approach is to use software environments. Moons identifies three types of software environments for learning programming [17]: a) micro-world based environments, such as Scratch [24], LOGO [25], Karel the Robot [26], Jeroo [27], Alice [28], Kit Lego Mindstorms [29] and CMOTION [30]; b) algorithm visualisation tools, such as Tango [31], Animal [32], Jawa [33], Jhavé [34], Alvis Live! [35] and VisBack [36]; and c) source code visualization and editing tools, such as DrJava [37], BlueJ [38], ProfessorJ [39], JGrasp [40], JIVE [41], JELiot 3 [42] and Ville [43]. Among the tools listed in paragraphs b and c, most are oriented to the OOP paradigm, as is the case of BlueJ, which is widely used in Java programming courses. It should be noted that object-oriented languages (such as Java), as opposed to other languages, entail a series of additional concepts such as classes, access modifiers, objects, inheritance, etc., which may be difficult to understand, but still have the advantage that to date they remain one of the most popular languages in professional environments, since they offer the possibility of developing applications for multiple platforms and technologies, which can be motivating for students who see their learning as a professional incentive.

Microworld-based tools promote a high level of motivation and a positive perception of learning programming [44]. These tools facilitate programming and learning through the visualized design and programming interface, because they allow users to program by manipulating program's elements graphically rather than specifying them textually [45]. These characteristics of graphic environments allow a practical learning process, based on the concept of learning by doing [22]. Regarding the tools for the visualization of algorithms, the students who use these tools decrease the semantic errors and keep the attention on the specific details of the behaviour of the algorithm [35]. Finally, visualization and editing tools for source code allow students to see the structure of programs at run and design time (through artifacts such as class diagrams), even in some cases allowing automatic code generation [17]. However, these types of tools have the limitation that the student has to deal with the syntactic complexity of programming language.

Unlike other tools, the solution presented in this article combines the characteristics of micro-world based environments with Algorithm visualization tools. This solution features a scaffolding mechanism [46], where the student is presented with small fragments of source code combined with graphical expressions to relate the solution phases of a problem to a specific textual syntax of a particular programming language. This way the student relates the syntactic expressions with graphic expressions according to their abilities, making them gradually master the syntax of the programming language they are using.

III. VILEP (VISUAL LEARNING OBJECT-ORIENTED PROGRAMMING)

This section describes the details of design and implementation of the tool, which has been developed using MDE (Model-driven engineering) techniques and reaches a specific instance through specific models and a series of transformations between models. The design is structured in three main components: a) a metamodel created with Eclipse Modeling Framework (EMF), which allows to model the visual construction of the program developed by the student; b) a graphic user interface designed with Sirius so that the student can interact with the tool visually; and c) a set of model transformation rules to perform the code generation through Acceleo, from the program built by the student in the graphic editor. The result is a plug-in that the student can install in Eclipse. To do this, the student must add the folders with the metamodels and the graphic editor of the tool, adding a new option to create VILEP projects where files with the extension ".javamodel" can be created for students to design their programs.

A. Definition of the Metamodel

This subsection describes the metamodel designed for the OOP domain, which allows to ignore the complexity of the syntax of a language. Figure 1 shows the meta-classes of the metamodel and their relationships. The meta-classes used to describe the parts of an object-oriented program are:

- *Explanation*: This meta-class represents the concept of commenting on a source code and allows the student to automatically insert a short description of the code they are creating from their design.
- *Project*: This meta-class represents the concept of a project created by the student.
- *Class*: This meta-class represents the classes that the student will implement in their project.
- *Method*: This meta-class represents the methods or functions of a class.
- *Object*: This meta-class groups the daughter meta-classes *Operator*, *Variable* and *Message*, which represent respectively the available mathematical operations (addition, subtraction, multiplication, division and remainder), the declarations of the variables together with the types of data, and the reading and output of information by console.
- *Argument*: This meta-class models the input or output arguments of the methods.

These components are related to each other: "*Project*" contains "*Class*" and this in turn contains "*Method*." Methods are related to other methods and may contain several "*Objects*" which represent mathematical operations, reading and writing by keyboard. In addition, "*Method*" can have "*Argument*," whose references allow methods to have local variables for their operations.

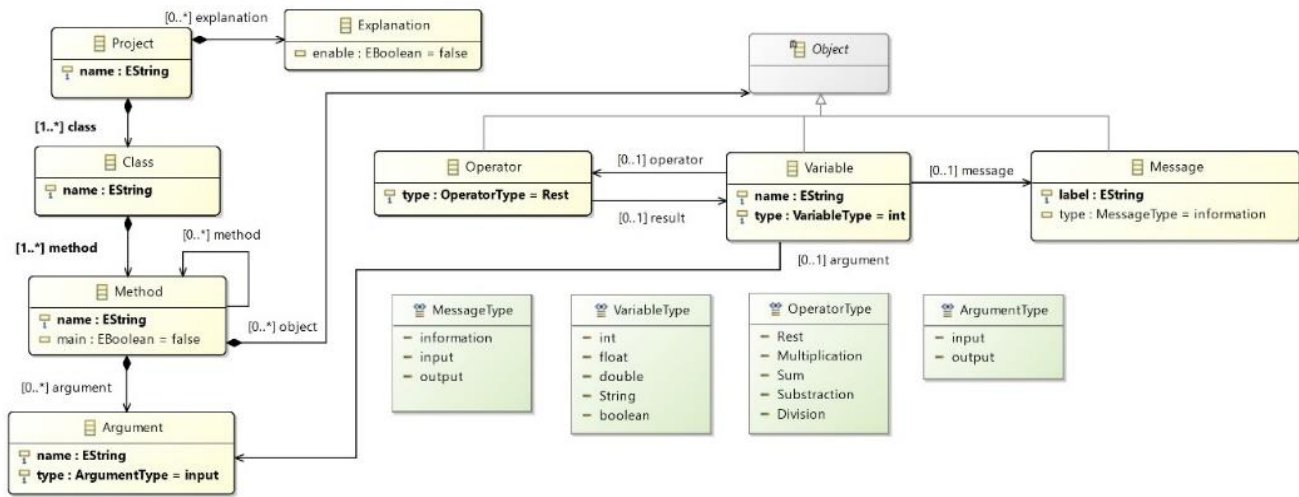


Fig. 1. Proposed metamodel for the visual tool

A. Model transformation

For the generation of the source code in Aceleo, we have implemented some transformation rules that sequence a series of basic tasks that must be done for any simple program and that we have specified in the transformation model. These tasks are listed below for the creation of Java files:

1. Creating the program files.
2. Declaring the required libraries for the methods.
3. Declaring the classes.
4. Declaring the methods.
5. Declaring the variables used.
6. Entering values.
7. Performing mathematical operations.
8. Displaying values.

These basic rules, which are learned at the beginning of the introductory programming courses, are graphically represented and the relationships that each of the components that end up forming a program are delimited. For example, methods are created only within classes, variables within methods, arithmetic operations receive numerical values stored in variables and their result is assigned to another variable. This way students focus on the concept of the task sequence that allows them to solve a problem, before facing the syntax of a language. With this tool students learn to perform arithmetic operations, data input and output by console, use of variables, method declaration, data types and class declaration.

The transformation rules defined, which transform the graphic expressions of the program designed by the student, generate a code (in this case in Java) that contains: a) Call to the *Scanner* Class of the *java.util* package for data entry by console, b) Declaration of a *Scanner* instance to store the information entered by console, c) Declaration of variables, d) Writing messages on screen with the *println()* method, e) Entry of values with the *Scanner* class *nextLine()* method and writing on screen messages and storage in a variable, f) Declaration of operations, and g) Writing on screen the results of operations with the method *println()*.

IV. VISUAL LEARNING OF PROGRAMMING WITH VILEP

This section describes the user interface of the tool, and its use from a teaching approach. VILEP allows the student to work with basic OOP concepts, such as classes, methods, arguments, and also with other programming concepts in general such as arithmetic expressions, assignments, input and output operations. Table 1 below summarises some of the differences between Eclipse and VILEP when used in the early stages of teaching programming.

TABLE 1. MAIN DIFFERENCES BETWEEN ECLIPSE AND VILEP

Eclipse	VILEP
The source code must be written manually.	Generating source code automatically.
Comments must be written by the programmer.	Didactic comments for the reading and interpretation of the source code.
Well integrated compilation and execution.	For the compilation and execution of the program a new document must be generated.
Use of textual syntax for all programming concepts.	Use of graphical syntax of the basic concepts of programming.
It does not have a Scaffolding mechanism.	Scaffolding mechanism that adjusts the syntax to the level of knowledge of the programmer.










The basic concepts of programming that the tool incorporates are represented by visual components available in a control palette, where the student can select and drag-drop on the editor (called Canvas) and make the design of a program. The editor visualizes the composition of the program at every moment showing some parts of the Java syntax and hiding others (the most complex ones) through visual icons. For example, the tool shows the complete syntax of a simple instruction in Java as an output operation: `"System.out.println("Enter to:");"` However, a more complex instruction such as an input operation on a variable such as: `"a=Integer.parseInt(Leer.nextLine());"` is represented visually, where a pencil symbol represents that it is a user typing operation on the keyboard and an "x" symbol in square brackets expresses that the result is assigned to a variable (in this case a variable called "a" of type int, see Mark B in Figure 2). As the student's activity is

not focused on language syntax, it presents less cognitive load, so the student can focus more on programming concepts during the creation of programs. However, the tool eliminates levels of abstraction and shows more detail of the language structures as the student advances in the handling of the syntax. The teacher adjusts to each problem or activity different levels of scaffolding according to different degrees of knowledge. Each student has an associated knowledge profile and the tool adjusts the scaffolding according to their profile. As the scaffolding progresses in the activities, the profile is adjusted. This way, VILEP adapts the scaffolding level for a progressive understanding of the student in the creation of a program for a specific programming language. The tool offers four main functionalities:

- Adding class, method and variable declarations to the program. These components are in the palette and the student drags them to compose the program.
- Perform console read and write (standard input and output operations).
- Perform basic mathematical operations (addition, subtraction, multiplication, division and remainder). It is possible to associate the variables with the values to operate and the variable to which the result of the operation is assigned.
- Describe invocations of selective and iterative control methods and structures.

Table 2 shows the components of VILEP and their description. Students can design basic algorithms in a graphic way without going into further details of a language syntax. The graphic designs generated by the student are interpreted with a reading flow from top to bottom and from left to right, which is translated by the tool and inserted in a ".java" file. However, some concepts such as the declaration of objects or inheritance have not been considered. Taking into account the meaning of the visual representations in Table 1, the program fragment in Figure 2 can be easily interpreted: a class called Operations is declared, which contains the "Main" method. Within this method, the following sequence of instructions is described: a message is written on the screen ("Enter A"), it is declared and reads the keyboard in the variable "a," then it does the same for the variable "b" and then it multiplies its two contents and the result is assigned to the variable "x" declared as *int*, then the variable "c," which was entered by keyboard, is added to "x" and the result is assigned in "y," finally the content of this variable is printed on the screen.

TABLE 2. MAIN COMPONENTS AND THEIR VISUAL REPRESENTATIONS

Component	Visual representation	Description
Class	 Class	Declaration of a class.
Method	 Method	Declaration of a method.
Argument	 Argument	Actual parameters of methods.
Variable	 Variable	Variable declaration.
Operator	 Operator	Arithmetic operator.
Message	 Message	Writing text by console.
Input	 Input	Reading from the keyboard.
Output	 Output	Writing a variable by console.
Connection	 Connection	Link between program components.

A. Teaching method

The aim of the tool is to reduce the cognitive workload of a language for students who are inexperienced in programming. For this purpose, the learning activities with the tool are developed with short work tasks, so that the student gains confidence to face more difficult problems, which will require more complicated language structures. The teaching approach is mainly based on the fact that students will use the tool to learn the principles of the basic syntax of the programming language. While using the tool's editor, they will visually observe how some lines of code are built and at the end they will observe the complete code. This allows the student to see immediately how the decisions they make in the design of the programs are directly reflected in a specific syntax of a programming language. In this context, the visual learning that is generated will allow students to gradually gain confidence in writing programs for a specific language.

The teaching methodology for the use of the VILEP tool can be broadly summarised in the following steps:

1. The teacher explains the basic programming concepts and briefly presents the syntax of Java. In addition, they must show the visual representation of the VILEP components of these concepts.
2. The teacher will propose a statement of a problem to be solved and will reflect with the students on the steps to be taken to solve it.
3. Subsequently, the students use VILEP autonomously to implement the program that solves the proposed problem. During this design process the tool will show fragments of source code associated with certain actions, and hide others. Finally, the tool will generate the source code it has implemented extended with explanatory comments.
4. The teacher will explain the doubts and the students will review and be able to execute the program they have obtained and verify the validity of its solutions.

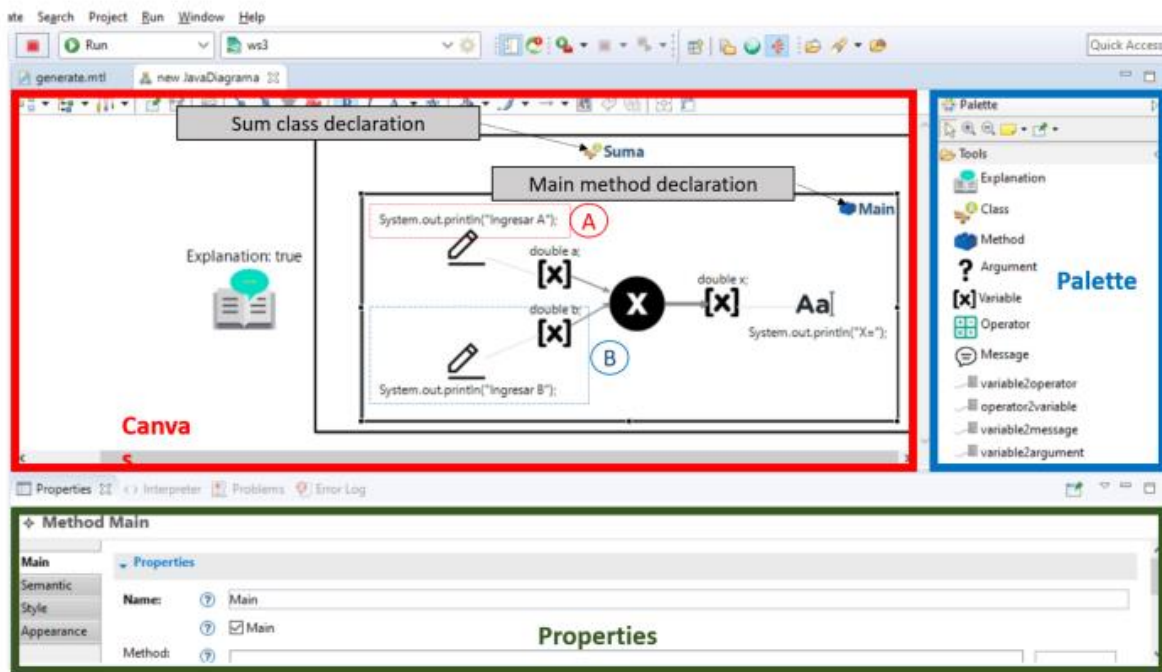


Fig. 2. Composition of a program with VILEP

I. CLASSROOM EXPERIENCE AND RESULTS

To demonstrate the validity of the tool in the teaching process, an exploratory study was carried out with students in the classroom. It is detailed below.

A. Teaching method

The experience with students aimed to validate whether the use of VILEP in an introductory programming course improves learning outcomes and the emotional state of the student during the learning process.

B. Sample

The sample selected was students from the first year of the Electronics and Automation degree at the University of the Armed Forces ESPE, in Quito, Ecuador, in the second semester of 2018, and was made up of 19 subjects (17 men and 2 women), who had no previous programming experience. This sample was randomly organized into two groups: experimental group (EG) and control group (CG).

C. Variables and instruments

The independent variable was the applied teaching tool: in the CG the classic teaching tool of a development environment was applied, while in the EG the VILEP tool was used. The dependent variables measured were the level of knowledge acquired and the positive and negative emotions experienced. For this purpose, in both groups, a pre-test of these variables was made at the beginning of the experience, and a post-test at the end. The instruments to measure these variables were two scales. Firstly, a knowledge scale with 6 multi-option items designed specifically for the experience. This scale raised questions about basic OOP concepts in which the student had to

interpret source code in Java. Secondly, a validated scale was used to measure emotions: Watson's PANAS [47]. The reason for using this scale is that it is already validated in the educational context and it allows to value the positive and negative emotions of the student in the learning task. The scale is composed of 20 terms (Table 3) that describe emotions of a positive or negative nature (10 of them positive and 10 negative). The students must evaluate how they feel for each of these emotional terms by means of a Likert scale with 5 options of response (not at all, very little, some, quite and a lot).

TABLE 3. PANAS EMOTIONS SCALE

Positive emotions terms		Negative emotion terms	
Interested	Decided	Disgusted	Tense
Disposed	Attentive	Guilty	Ashamed
Animated	Active	Fearful	Nervous
Enthusiastic	Energetic	Angry	Uneasy
Proud	Inspired	Irritated	Scared

D. Method

The experience was carried out with the students of the Introduction to Engineering course, and one of the learning outcomes is to study the basic concepts of programming. To this end, 2 of the 6 sessions of the course were used to achieve the learning outcome. The experience began with an explanation to the students of the objectives of the activity and requesting their consent to participate (100% of the students participated). Then, the participants were randomly organized in two groups: a) EG (Experimental Group), a group made up of 10 participants who used the VILEP tool, and b) CG (Control Group), a group made up of 9 participants who had the usual teaching method using the Eclipse environment. While the tool was used in an exploratory way for this experience of some basic OOP

concepts, it could be used for the rest of the course concepts.

Figure 3 shows the development of the experience. Once the groups were constituted, the intervention began with an initial assessment of the students' knowledge and emotional state. Then, the teacher (the same in both groups) explained the theoretical foundations of OOP (classes, methods and attributes) and the basic syntax of Java (class and attribute statements, arithmetic operators, input and output), using the specific tool for each group. Later, in the EG, they used the VILEP tool and the CG used Eclipse to develop the first

program with the advice of the teacher. Next, both groups did several tests implementing a basic Java program, the EG using VILEP by visual programming and the CG using Eclipse with classic textual programming. Figure 4 shows a screenshot of a program developed by CG students using Eclipse, while Figure 2 shows a program developed by EG students using VILEP. Finally, knowledge and emotions were re-evaluated after the completion of the task. The complete planning and its time distribution for each group is shown in Table 4.

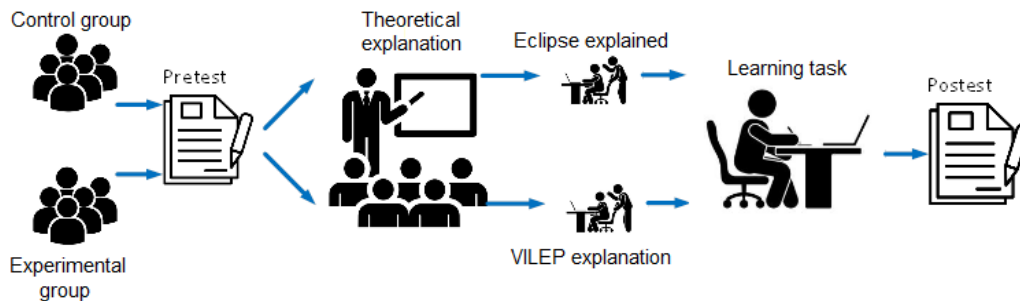


Fig. 3. Methodology of the classroom experience

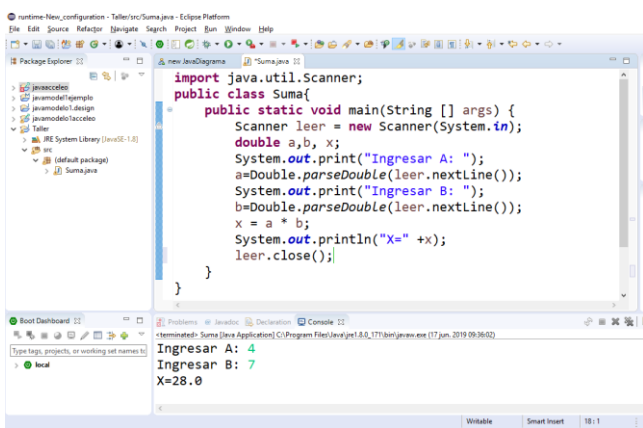


Fig. 4. Control group learning task

A. Experimental results

For the comparison of the results obtained in both groups, two analyses were carried out: a) a comparison of hypotheses of means equality of the measured variables and b) a comparison of the error rates in the learning task. In order to carry out the comparison of means equality, the following statistical variables have been defined:

- KNOWLEDGE_PRE: These are the results of the level of knowledge at the beginning of the experience.
- KNOWLEDGE_POS: This is the knowledge of the students after the experience.
- POSITIVE_EMOTIONS_PRE: These are the positive emotions of the students at the beginning of the experience.
- POSITIVE_EMOTIONS_POS: It shows students'

positive emotions after the experience.

- NEGATIVE_EMOTIONS_PRE: It measures students' negative emotions at the beginning of the experience.
- NEGATIVE_EMOTIONS_POS: These are the negative emotions of the students at the end of the experience.
- NEGATIVE_PERCEPTIONS_TAR: It measures the negative perceptions of the students after the experience.
- POSITIVE_PERCEPTIONS_TAR: It shows students' positive perceptions of the experience.

Table 5 shows the descriptive statistics comparing both groups. Table 5 shows that the level of knowledge at the end of the experience is higher in the group that used VILEP than in those that did not (variable KNOWLEDGE_POS=3.8 vs. 3.67). Similarly, it can also be seen in this same table that the positive emotions were higher at the end of the experience in the students who used VILEP than in the students who used Eclipse (variable POSITIVE_EMOTIONS_POS=38.8 vs. 37.78). However, it was also identified that negative emotions were slightly higher in the experimental group than in the control group. In order to check if these differences are statistically significant or not, a contrast test of means was made. To do so, firstly, it was identified which variables followed a normal distribution, applying the *Shapiro Wilk test* ($p=0.05$) when dealing with small samples.

TABLE 4. TEMPORALITY OF THE EXPERIENCE

Phase	Experimental Group	Control group
Presentation and group formation	15"	15"
Pre-test performance (knowledge and emotions)	30"	30"
Explanation of OOP and Java concepts with VILEP	1h	-
Explanation of OOP and Java concepts with Eclipse	-	1h
Use of VILEP with teacher assistance	30"	-
Using Eclipse with Teacher Assistance	-	30"
Performing autonomous program implementation tasks VILEP	30"	-
Performing Standalone Program Deployment Tasks Eclipse	-	30"
Post-test performance (knowledge and emotions)	30"	30"
TIME OF EXPERIENCE	3h 15"	3h 15"

In this study it was found that all followed a normal distribution except the variables POSITIVE_EMOTIONS_POS and POSITIVE_PERCEPTIONS_TAR. For these two variables, the mean comparison was performed by the non-parametric *Mann-Whitney* test, while for the rest of the variables the *t-Student* test was performed (both with a significance level $p=0.05$). The hypothesis contrast in both tests indicated that there was no difference between the means in any of the variables. Therefore, although no significant difference was found, it can be stated that there is a trend of improvement in the level of knowledge and in the positive emotions of the VILEP use against the Eclipse use, although there is a trend of worsening in the negative emotions.

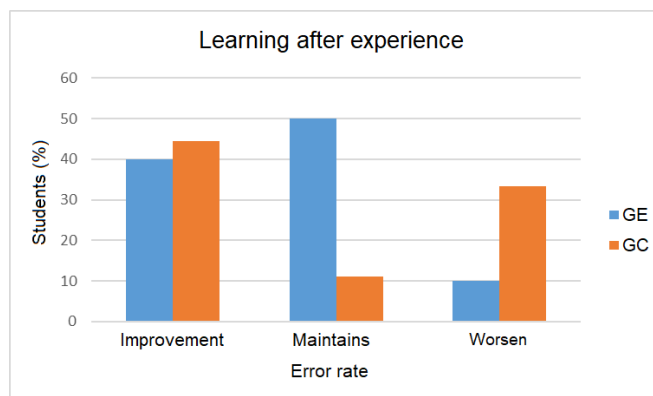


Fig. 5. Learning outcomes according to error rate

On the other hand, analysing the number of students who increased their positive emotions in one group and in another, it can be seen that in the group that used VILEP, 60% of the students increased their positive emotions during the completion of the task, while in the control group it was only 44.4% of the students. Therefore, it can be understood that the use of visual programming with VILEP fosters positive emotions of some students in the process of programming learning, although it should not be neglected that the use of the tool did not decrease negative emotions as much as it did in the control group (40% of GE vs 66.6% of GC). These results could be related to the side effect of the cognitive burden that the use of visual representations in programming learning entails [48].

As indicated above, in a second phase of result analysis, the pre-test and post-test error rates were reviewed for each group. In the EG at the end of the experience, 40% of the students improved their results, 50% maintained their results and 10% of the students increased their errors. In the CG, 44.5% of the students improved their results, 11.1% maintained the same level and 33.4% increased their errors. As can be seen, almost half of the students in the two groups improved their scores (44.5% of the CG vs. 40% of the EG) however, in the control group there was a significant increase of students with more errors compared to the experimental group (33.4% of the CG vs. 10% of the EG). Figure 5 graphically shows these data, where it can be seen how students who used VILEP either improved or maintained their results, and very few worsened them, compared to those who used Eclipse, where there was a high number of students who made more errors in the post-test. Hence, based on these data it seems that the use of visual programming with the VILEP tool results in students making fewer errors in coding. The authors think that this could be due to the fact that visual programming with VILEP allows the student to focus only on the concepts and structures of programming during the learning process, making fewer mistakes later on when applying them to programme coding. This would explain why students who used Eclipse made more coding errors, since in the learning process they had to work with the concepts and programming structures, together with the difficulty of language syntax.

TABLE 5. DESCRIPTIVE STATISTICS OF THE TWO GROUPS

	Experimental group (GE) (N=10)				Control group (GC) (N=9)			
	Minimum	Maximum	Half	Standard deviation	Minimum	Maximum	Half	Standard deviation
KNOWLEDGE_PRE	1	6	3,50	1,581	0	6	2,89	1,764
KNOWLEDGE_POS	2	6	3,80	1,549	1	6	3,67	1,658
POSITIVE_EMOTIONS_PRE	33	46	38,90	3,814	28	44	37,22	5,019
POSITIVE_EMOTIONS_POS	28	44	38,80	4,417	29	47	37,78	6,360
NEGATIVE_EMOTIONS_PRE	10	22	14,50	4,275	10	22	14,67	3,841
NEGATIVE_EMOTIONS_POS	10	22	13,70	4,029	10	17	12,89	2,619
NEGATIVE_PERCEPTIONS_TAR	1,2	2,8	2,060	,5337	1,2	3	1,911	,5754
PERCEPTIONS_POSITIVAS_TAR	4	5	4,70	,483	4	5	4,78	,441

II. CONCLUSION

In computer engineering and related degrees, introductory programming subjects have low learning outcomes and low pass rates. One of the reasons that generates these bad results is the difficulty that these inexperienced students have in understanding and properly handling the syntax of programming languages. These difficulties end up generating a lack of motivation in the student, because not only must they face the challenge of understanding fundamental programming concepts, but also

1 must learn the programming language associated with those
 2 concepts. This article presents the VILEP tool, which
 3 provides a visual programming editor that abstracts the
 4 complexity of language use and focuses on the concepts of
 5 programming. The tool implements a mechanism of
 6 Scaffolding that allows the student to visualize or not the
 7 most complex parts of the syntactic structures of
 8 programming languages. The mechanism adds visual
 9 representations and combines them with fragments of
 10 source code, so that as the student assimilates the syntax of
 11 the language, the tool decreases the level of abstraction and
 12 shows more detail of the syntactic structures until the
 13 student ends up developing the source code directly. This
 14 first exploratory study was structured with the first year
 15 students of the Electronics and Automation degree. Two
 16 work groups were organized to develop a program in Java:
 17 one of them used the proposed VILEP tool and the other
 18 worked with the Eclipse development environment through
 19 text programming (a common tool used in computer science
 20 subjects). A knowledge scale was used to measure learning
 21 outcomes and the PANAS scale to measure students'
 22 emotions during the learning process. It can be concluded
 23 that the proposed visual programming tool considerably
 24 reduced the number of students who at the end of the
 25 experience made mistakes with the programming language
 26 with respect to the group that did not use the tool (23.4%
 27 fewer students). In addition, it was found that more students
 28 experienced positive emotions during the programming task
 29 when using the VILEP tool (60% of the students in the
 30 group) than when not using it (44.4% of the group).

31 Based on these results, some lines of future work are
 32 opened. The analysis of the results is an exploratory
 33 analysis, so it is necessary to replicate new experiences in
 34 other universities with a deeper statistical analysis and with
 35 a greater number of participants, to confirm whether the
 36 improvements in learning outcomes and emotions found are
 37 statistically significant. In addition, studies should be
 38 carried out that analyze the correlations between learning
 39 outcomes and emotions during the programming learning
 40 process. On the other hand, it is necessary to incorporate
 41 new features such as reverse engineering to graphically
 42 represent programs made in text form, expansion of
 43 functions for OOP and web resources of the project with
 44 additional information so that the tool can be used by other
 45 users.

46 ACKNOWLEDGMENT

47 This work has been financed thanks to iProg from
 48 MINECO (ref. TIN2015-66731-C2-1-R) and e-Madrid-CM
 49 (ref. P2018 / TCS-4307) with FSE and FEDER funds.

50 REFERENCES

- 51 [1] H. Amer and A. Ain, "Smart – Learning Course Transformation
 52 for an Introductory Programming Course," *2017 IEEE 17th Int.
 53 Conf. Adv. Learn. Technol.*, pp. 463–465, 2017.
- 54 [2] A. Forte and M. Guzdial, "Motivation and Nonmajors in
 55 Computer Science : Identifying Discrete Audiences for
 56 Introductory Courses," vol. 48, no. 2, pp. 248–253, 2005.
- 57 [3] L. McIver and D. Conway, "Seven Deadly Sins of Introductory
 58 Programmming Language Design Linda," *Notes Queries*, pp.
 59 309–316, 1996.
- [4] E. Kaila, M. Laakso, T. Rajala, A. Mäkeläinen, and E. Lokkila,
 "Technology-Enhanced Programming Courses for Upper
 Secondary School Students," pp. 683–688, 2018.
- [5] R. Mahmudur and R. Paudel, "Preliminary Experience and
 Learning Outcomes by Infusing Interactive and Active Learning
 to Teach an Introductory Programming Course in Python," 2018.
- [6] C. C. W. Hulls, A. J. Neale, B. N. Komalo, V. Petrov, and D. J.
 Brush, "Interactive Online Tutorial Assistance for a First
 Programming Course," vol. 48, no. 4, pp. 719–728, 2005.
- [7] M. Schmidt, V. Benzing, A. Wallman-Jones, M. F. Mavilidi, D.
 R. Lubans, and F. Paas, "Embodied learning in the classroom:
 Effects on primary school children's attention and foreign
 language vocabulary learning," *Psychol. Sport Exerc.*, vol. 43,
 pp. 45–54, 2019.
- [8] O. Debbi, M. Paredes-Velasco, and J. A. Velazquez-Iturbide,
 "Influence of Pedagogic Approaches and Learning Styles on
 Motivation and Educational Efficiency of Computer Science
 Students," *Rev. Iberoam. Tecnol. del Aprendiziz.*, vol. 11, no. 3,
 pp. 213–218, 2016.
- [9] K. Willey and A. Gardner, "Collaborative learning frameworks
 to promote a positive learning culture," *Proc. - Front. Educ.
 Conf. FIE*, pp. 1–6, 2012.
- [10] L. P. Nelson and M. L. Crow, "Do Active-Learning Strategies
 Improve Students' Critical Thinking?," *High. Educ. Stud.*, vol. 4,
 no. 2, pp. 77–90, 2014.
- [11] X. Basogain-Olabe, M. Á. Olabe-Basogain, and J. C. Olabe-
 Basogain, "Pensamiento Computacional a través de la
 Programación: Paradigma de Aprendizaje," *Rev. Educ. a
 Distancia*, vol. 46, no. 46, 2015.
- [12] O. Aktunc, "A Teaching Methodology for Introductory
 Programming Courses using Alice," *Int. J. Mod. Eng. Res.*, vol.
 3, no. 1, pp. 350–353, 2013.
- [13] C. Dumitrescu, R. L. Olteanu, L. M. Gorghiu, and G. Gorghiu,
 "Using virtual experiments in the teaching process," vol. 1, no.
 1, pp. 776–779, 2009.
- [14] M. Kölling, "The Greenfoot Programming Environment," *ACM
 Trans. Comput. Educ.*, vol. 10, no. 4, pp. 1–21, 2010.
- [15] J. van Niekerk and P. Webb, "The effectiveness of brain-
 compatible blended learning material in the teaching of
 programming logic," *Comput. Educ.*, vol. 103, pp. 16–27, 2016.
- [16] G. M. M. Bashir and A. S. M. L. Hoque, "An effective learning
 and teaching model for programming languages," *J. Comput.
 Educ.*, vol. 3, no. 4, pp. 413–437, 2016.
- [17] J. Moons and C. De Backer, "The design and pilot evaluation of
 an interactive learning environment for introductory
 programming influenced by cognitive load theory and
 constructivism," *Comput. Educ.*, vol. 60, no. 1, pp. 368–384,
 2013.
- [18] M. Koorsse, C. Cilliers, and A. Calitz, "Programming assistance
 tools to support the learning of IT programming in South African
 secondary schools," *Comput. Educ.*, vol. 82, pp. 162–178, 2015.
- [19] P. Tshering, D. Lhamo, L. Yu, and A. Berglund, "How Do First
 Year Students Learn C Programming in Bhutan?," *Proc. - 5th
 Int. Conf. Learn. Teach. Comput. Eng. LaTiCE 2017*, pp. 25–29,
 2017.
- [20] C. J. Olelewe and E. E. Agomuo, "Effects of B-learning and F2F
 learning environments on students' achievement in QBASIC
 programming," *Comput. Educ.*, vol. 103, pp. 76–86, 2016.
- [21] K. J. Harms, J. Chen, and C. Kelleher, "Distractors in parsons
 problems decrease learning efficiency for young novice
 programmers," *ICER 2016 - Proc. 2016 ACM Conf. Int. Comput.
 Educ. Res.*, pp. 241–250, 2016.
- [22] J. M. Su and T. W. Lin, "Building a Simulated Blockly-Arduino-
 Based Programming Learning Tool: A Preliminary Study," *Proc.
 - 2018 7th Int. Congr. Adv. Appl. Informatics, IIAI-AAI 2018*, pp.
 378–381, 2018.
- [23] D. Saito, H. Washizaki, and Y. Fukazawa, "Work in progress: A
 comparison of programming way: Illustration-based
 programming and text-based programming," *Proc. 2015 IEEE
 Int. Conf. Teaching, Assess. Learn. Eng. TALE 2015*, no.
 December, pp. 220–223, 2016.
- [24] S. Uludag, M. Karakus, and S. W. Turner, "Implementing

- IT0/CS0 with scratch, app inventor for android, and lego mindstorms,” *SIGITE'11 - Proc. 2011 ACM Spec. Interes. Gr. Inf. Technol. Educ. Conf.*, pp. 183–189, 2011.
- [25] D. M. Kurland and R. D. Pea, “Children’s Mental Models of Recursive Logo Programs,” *J. Educ. Comput. Res.*, vol. 1, no. 2, pp. 235–243, 1985.
- [26] K. Dai, Y. Zhao, and R. Chen, “Research and practice on constructing the course of programming language,” *Proc. - 10th IEEE Int. Conf. Comput. Inf. Technol. CIT-2010, 7th IEEE Int. Conf. Embed. Softw. Syst. ICESS-2010, ScalCom-2010*, no. Cit, pp. 2033–2038, 2010.
- [27] B. Dorn, “Jeroo : A Tool for Introducing Object-Oriented Programming Northwest Missouri State University,” *Control*, pp. 201–204, 2003.
- [28] S. Cooper, W. Dann, and R. Pausch, “Teaching objects-first in introductory computer science,” *SIGCSE Bull. (Association Comput. Mach. Spec. Interes. Gr. Comput. Sci. Educ.)*, pp. 191–195, 2003.
- [29] S. H. Kim and J. W. Jeon, “Introduction for freshmen to embedded systems using LEGO mindstorms,” *IEEE Trans. Educ.*, vol. 52, no. 1, pp. 99–108, 2009.
- [30] S. L. Finkelstein, A. Nickel, L. Harrison, E. A. Suma, and T. Barnes, “cMotion: A new game design to teach emotion recognition and programming logic to children using virtual humans,” *Proc. - IEEE Virtual Real.*, pp. 249–250, 2009.
- [31] J. T. Stasko, “Tango: A Framework and System for Algorithm Animation,” *Computer (Long. Beach. Calif.)*, vol. 23, no. 9, pp. 27–39, 1990.
- [32] G. Rbling, M. Sch, B. Freisleben, and D.- Siegen, “The ANIMAL Algorithm Animation Tool,” pp. 37–40, 2000.
- [33] A. Akingbade, T. Finley, D. Jackson, P. Patel, and S. H. Rodger, “JAWAA: Easy web-based animation from CS 0 to advanced CS courses,” *SIGCSE Bull. (Association Comput. Mach. Spec. Interes. Gr. Comput. Sci. Educ.)*, pp. 162–166, 2003.
- [34] T. L. Naps, J. R. Eagan, and L. L. Norton, “JHAVE - an environment to actively engage students in Web-based algorithm visualizations,” *SIGCSE Bull. (Association Comput. Mach. Spec. Interes. Gr. Comput. Sci. Educ.)*, pp. 109–113, 2000.
- [35] C. D. Hundhausen and J. L. Brown, “Designing, visualizing, and discussing algorithms within a CS 1 studio experience: An empirical study,” *Comput. Educ.*, vol. 50, no. 1, pp. 301–326, 2008.
- [36] C. Lacave, J. Á. Velázquez-Iturbide, M. Paredes-Velasco, and A. I. Molina, “Analyzing the influence of a visualization system on students’ emotions: An empirical case study,” *Comput. Educ.*, vol. 149, no. January, 2020.
- [37] E. Allen and B. Stoler, “Dr Java : A lightweight pedagogic environment for Java,” 2002.
- [38] M. C. Jadud, “A First Look at Novice Compilation Behaviour Using BlueJ,” *Comput. Sci. Educ.*, vol. 15, no. 1, pp. 25–40, 2005.
- [39] K. E. Gray and M. Flatt, “ProfessorJ : AA gradual introduction to Java through language levels,” *Proc. Conf. Object-Oriented Program. Syst. Lang. Appl. OOPSLA*, pp. 170–177, 2003.
- [40] J. H. Cross and D. Hendrix, “Workshop jGRASP: An integrated development environment with visualizations for teaching Java in CS1, CS2, and beyond,” *Proc. - Front. Educ. Conf. FIE*, p. 1, 2006.
- [41] P. V. Gestwicki and B. Jayaraman, “JIVE: Java interactive visualization environment,” *Proc. Conf. Object-Oriented Program. Syst. Lang. Appl. OOPSLA*, pp. 226–227, 2004.
- [42] A. Moreno and M. S. Joy, “Jeliot 3 in a Demanding Educational Setting,” *Electron. Notes Theor. Comput. Sci.*, vol. 178, pp. 51–59, 2007.
- [43] T. Rajala, M.-J. Laakso, E. Kaila, and T. Salakoski, “VILLE -- A Language-Independent Program Visualization Tool,” *Seventh Balt. Sea Conf. Comput. Educ. Res. (Koli Call. 2007)*, vol. 88, no. January 2016, pp. 151–159, 2007.
- [44] B. Kaučić and T. Asič, “Improving introductory programming with Scratch?,” *MIPRO 2011 - 34th Int. Conv. Inf. Commun. Technol. Electron. Microelectron. - Proc.*, pp. 1095–1100, 2011.
- [45] J. M. Su and S. J. Wang, “A Web-Based learning activity integrated with scratch tool to support programming learning,” *Ubi-Media 2017 - Proc. 10th Int. Conf. Ubi-Media Comput. Work. with 4th Int. Work. Adv. E-Learning 1st Int. Work. Multimed. IoT Networks, Syst. Appl.*, pp. 1–4, 2017.
- [46] C. H. Chen and V. Law, “Scaffolding individual and collaborative game-based learning in learning performance and intrinsic motivation,” *Comput. Human Behav.*, vol. 55, pp. 1201–1212, 2016.
- [47] A. Watson, D., Clark, L., & Tellegen, “Development and validation of brief measures of positive and negative affect: the PANAS scales,” *J. Pers. Soc. Psychol.*, vol. 54, no. 6, pp. 1063–1070, 1988.
- [48] P. Crescenzi, A. Malizia, M. C. Verri, P. Diaz, and I. Aedo, “On Two Collateral Effects of Using Algorithm Visualizations,” *Br. J. Educ. Technol.*, vol. 42, no. 6, pp. 145–147, 2011.

Darwin Alulema received the degrees of Engineer in Electronics and Telecommunications (Ecuador-2006), Lawyer of the Courts and Tribunals of the Republic (Ecuador-2011), Master in Teleinformatics and Computer Networks (Ecuador-2009), Master in Multimedia Applications (Spain-2015) and Master in Civil Law and Civil Procedure (Ecuador-2016). Since 2007 he has been a Professor at the University of the Armed Forces, in Ecuador.

Maximiliano Paredes-Velasco received the degree of Ph.D. in computer science from the University of Castilla-La Mancha, Spain, in 2006. He is currently a Professor at the Rey Juan Carlos University. His research interests include different fields of computer-supported learning (collaborative learning, active learning, and mobile learning) and human-computer interaction. He is the author of numerous articles and several international conference papers. His research focuses on collaborative learning, ubiquitous computing, and human-computer interaction.