MoCAS: A Mobile Collaborative Tool for Learning Scope of Identifiers in Programming Courses

Luis Miguel Serrano-Cámara, Maximiliano Paredes-Velasco, J. Ángel Velázquez-Iturbide, Carlos-María Alcover and Mª Eugenia Castellanos

Universidad Rey Juan Carlos

C/ Tulipán s/n, Móstoles, 28933 Madrid, Spain

luismiguel.serrano.camara@urjc.es, maximiliano.paredes@urjc.es, angel.velazquez@urjc.es, carlosmaria.alcover@urjc.es, maria.castellanos@urjc.es

Abstract

This article presents an instructional framework for collaborative learning, called CIF and aimed at the analysis level of Bloom's taxonomy, as well as a mobile collaborative tool called MoCAS that supports CIF. MoCAS is aimed at the domain of scope of identifiers in programming learning, which is a topic present in programming courses in engineering studies. The specification and development of MoCAS were explicitly driven by pedagogical goals and by the atomic actions declared in CIF as simple items of collaborative activities. Furthermore, CIF and MoCAS were evaluated in an actual educational context with respect to students' performance and motivation. Students using CIF and MoCAS obtained statistically significant higher grades than students studying in an individual or collaborative basis but not using MoCAS. In addition, we measured statistically significant measures indicating that students instructed with CIF and MoCAS were more motivated than students instructed collaboratively but not using CIF or MoCAS. In addition to CIF and MoCAS, and the evaluation results, the experiences here reported exemplify several software engineering practices: the design of an educational system based on knowledge of the target domain (namely, Bloom's taxonomy) and the evaluation of users' satisfaction (mainly, students' motivation).

Keywords

CSCL; Bloom's taxonomy; computer programming; learning performance; motivation.

1. Introduction

Programming is one of the basic areas in engineering education [1] as well as a core area in the computing disciplines, including Computer Engineering [2] and Software Engineering [3]. Learning programming is a demanding task. Therefore, many efforts have been dedicated to teaching programming more effectively since the eighties [4]. Two of the mainstream areas of research have been innovative educational methods [5] and new learning tools [6]. However, it is uncommon to have these efforts driven by clear and explicit learning goals.

Our hypothesis is that the use of explicit pedagogical goals increases the quality and effectiveness of learning tools. From the point of view of a software engineer, tool requirements can more easily be identified and consequently their specification can more easily be stated. From the point of view of an instructor, instructional activities can be checked for alignment with the educational goals supported by the system. We also need to take into consideration the instructional approach that the learning tool would support. There are a number of successful active learning approaches [7], such as problem-based learning [8] or collaborative learning [9]. Depending on the instructional approach, different instructional activities must be designed, scheduled and performed.

In this article we present three contributions regarding collaborative learning tools (i.e. CSCL systems). Firstly, we specified a collaborative instructional framework called CIF in terms of the well-known Bloom's taxonomy [10]. Bloom's taxonomy establishes a hierarchy of six levels of increasing degree of student's cognitive mastery, where every level assumes that the student has achieved certain degree of mastery in the lower levels. CIF was designed to support the analysis level of Bloom's taxonomy. This level is especially appropriate for collaborative learning because it fits well with discussion activities. Lower levels in Bloom's taxonomy, such as the comprehension level, fit better (but not necessarily) with individual activities. Higher levels in Bloom's taxonomy, such as the synthesis or

evaluation levels, imply a higher level of expertise from students. Secondly, MoCAS was built to support the collaborative activities identified in CIF in the domain of scope of identifiers in programming learning, which is a topic present in programming courses in engineering studies. MoCAS supports the use of personal computers and laptops but also mobile devices, in order to foster their adoption, use and acceptance by students. Finally, we conducted a comprehensive evaluation of MoCAS with respect to two criteria, namely educational efficiency and motivation of students, each one involving four groups of students.

The structure of the article follows. In the second section we present the collaborative instruction framework CIF and in the third section we present the MoCAS tool. Section four describes the two evaluations conducted of MoCAS with respect to learning efficiency and motivation, with statistically significant positive results. Sections five and six analyze related tools and contain a brief discussion regarding software engineering, respectively. Finally, we summarize our conclusions in the seventh section.

2. CIF – Collaborative Instructional Framework

The Collaborative Instructional Framework (CIF) is a collaborative instructional method conceived to design learning activities aimed at the acquisition of analysis skills. In the first subsection we present CIF in its general formulation and in the second subsection we present its instantiation to instruct a specific topic of programming, namely scope of identifiers in procedural languages.

2.1. The CIF Collaborative Instructional Framework

CIF provides a framework to design collaborative instructional activities aimed at the analysis level of Bloom's taxonomy [10]. According to Bloom, at the analysis level "the student should be able to distinguish, classify and relate hypothesis and evidences of the information given, as well as decomposing a problem into its parts" [10]. The analysis level distinguishes 16 specific analysis goals clustered into 3 groups:

- Analysis of elements. The simplest analysis tasks that can be addressed are the identification of elements about which information is given. This identification skill can be elaborated according to 3 goals. In this article, we focus on this group of goals, thus we enumerate them explicitly: (a) ability to recognize unstated assumptions, (b) ability to distinguish facts from hypothesis and from normative statements, and (c) ability to distinguish a conclusion from statements which support it.
- Analysis of relationships between elements. This group consists of 8 goals, aimed at identifying the relationships between elements, the relevance of facts to validate an assessment, as well as the existence of cause-effect relationships between facts.
- Analysis of organizational principles. It consists of 5 goals that intend to develop students' skill to analyze the influence of an author's opinion, of his/her background or of the state of the art on the facts presented to students in a task statement.

The CIF instructional framework supports the 16 analysis goals of Bloom's taxonomy by defining 16 guides, called Generic Cards (GC). A GC guides the instructor in designing, implementing and evaluating collaborative analysis activities. We identify a particular GC by referring to it by its corresponding number in the list of analysis goals given by Bloom, i.e. GC*x*, being *x* a natural number ranging from 1 to 16.

Each GC consists of three sections:

1. Task description. It states an intended pedagogical (analysis) goal.
2. Activities. It consists of a set of collaborative activities, adequately arranged as a sequence in order to achieve the pedagogical goal.
3. Evaluation. It suggests evaluation methods adequate to assess whether the pedagogical goal has been achieved.

The instructor wishing to use CIF will first select the analysis goal to achieve and its associated GC. Then, the instructor must adapt the GC to the domain of study, resulting in a Domain Card (DC). Once a DC is created, the instructor will follow the detailed steps stated in the DC card for his/her instruction. Figure 1 illustrates this procedure.

In order to make card instantiation easier, CIF defines a set of atomic actions that can be used to implement any kind of collaborative instruction in the class (see Table 1 for a subset). Consequently, the educational task described in a

GC or DC can be performed by means of a sequence of atomic actions. In the next subsection we give an example of a DC for the domain of scope of identifiers.

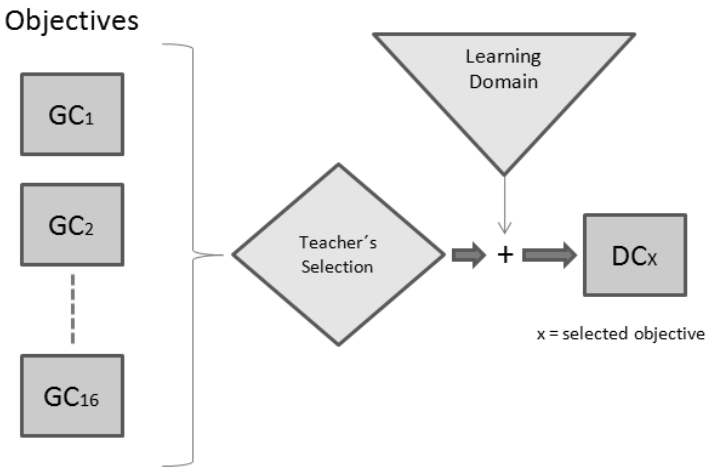

Figure 1. Procedure for instructing with CIF

Table 1. Atomic actions used in GC1 and DC1

Code	Description
AA1	Form groups of students
AA2	Deliver the problem statements to the groups
AA3	Perform an action appropriate to the problem statement
AA4	Exchange problem statements among the groups
AA5	Display all the answers given by groups to the same problem statement
AA6	Present and defend by each group its answers to the class
AA7	Debate raising disagreements
AA8	Discuss final answers
AA9	Instructor's mediation

2.2. Cards for the analysis of scope of identifiers

In the article we focus on the first pedagogical goal regarding the analysis level of Bloom's taxonomy, namely "the ability to recognize unstated assumptions". GC1 has been adapted to the topic of scope of identifiers in procedural languages, resulting in DC1. Therefore, DC1 is aimed at the "ability to distinguish and identify the scope of identifiers in a procedural programming language".

The structure of DC1 is shown in Table 2 (GC1 is not explicitly shown in another table because of its great similarity to DC1). Notice that some activities and their corresponding atomic actions have been grouped into larger activities. Strictly, we should separate them but this grouping will allow making the presentation of MoCAS easier in the next section. We do not include here the evaluation part of DC1 because we have focused on the task and activities sections of CIF, thus the evaluation section is not implemented in the current version of MoCAS. Guidelines about assessment of collaborative activities can be used as a basis for a future extension [11].

Table 2. DC1 – Domain Card for objective 1 instantiated to the domain of identifier scope in a procedural programming language

Task description	
Each exercise is a short program coded in Pascal, containing the main program and different subprograms with several identifiers. Several exercises are given to a class, so that each group is given two exercises and each exercise is solved by at least two groups. The aim of this task is to practice the ability to distinguish and identify the scope of identifiers occurring in the different parts of code. The students must create collaboratively a table declaring the scope of each identifier.	
Description of activities	**Atomic actions**
1. Different groups of 4 students are formed. Each group is given a problem statement (i.e. to identify the scope of the identifiers contained in a given program). The statement contains a piece of source code in Pascal and a table area where the solution must be written (i.e. a scope table).	AA1 AA2
2. Each group analyzes the source code contained in their problem.	AA3
3. Each group is given a second problem statement. Each group analyzes the source code contained in their second problem.	AA4 AA3
4. Once each problem code has been analyzed by at least two groups, the answers are clustered and displayed to the class.	AA5
5. The different answers are presented by members of their corresponding groups, paying special attention to the elements where there is no consensus.	AA6
6. A debate about disagreements is held and moderated by the instructor.	AA7 AA8 AA9

CIF constitutes a collaborative learning framework that can be used without computer support. However, instructors and students may obtain the highest instructional benefits by using a CSCL system to support CIF; therefore we developed the MoCAS system. Actually, we have evaluated both uses of CIF (with and without MoCAS support), as we report in section 4. First, we describe MoCAS in section 3.

3. The MoCAS system

The Mobile Collaborative Argument Support (MoCAS) system is a collaborative system developed to support and conduct collaborative activities according to CIF. More specifically, MoCAS was designed to support the learning of the topic scope of identifiers in procedural programming languages, as stated in the DC1 defined in Subsection 2.2. In its current state of implementation, MoCAS supports several procedural programming languages (Pascal, C and Java). MoCAS is supported in a number of computing devices, including personal computers, laptops and mobile devices.

MoCAS is based on a client/server architecture. The server controls all the contributions done by each member of the group. Each client has a user interface adapted to the device constraints, supporting mobile devices, PCs and laptops.

MoCAS was built under the Microsoft .Net Framework on the server site, and under the .Net Compact Framework on the mobile devices or .Net Framework on PCs with Windows Presentation Foundation to enrich the interface. The MySQL database engine was used to store information and, to promote further functionality, communication was based on WebServices by using SOAP protocol.

The hardware requirements to use the tool are: (1) a personal computer running any Windows version to install the MoCAS server, (2) mobile devices running Windows Mobile o PCs and laptops running Windows to install MoCAS clients, and (3) a Wireless Access point to implement the wireless net.

Figure 2 outlines the role of MoCAS in the collaborative instruction conceived by CIF.

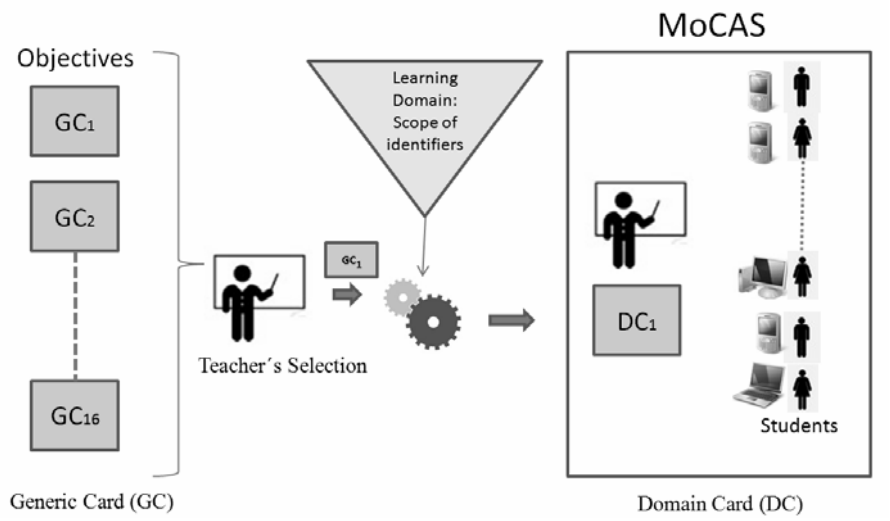

Figure 2. Procedure for instructing with CIF supported by MoCAS for the domain of scope of identifiers

MoCAS supports the complete instruction of the domain card DC1 (recall that it is aimed at learning the scope of identifiers in procedural programming languages). In this section, we present the main features of MoCAS by following the major steps in DC1 instruction. There are six steps corresponding to DC1 activities, as grouped in Table 2 above, plus a previous step for environment creation:

0. **Environment creation.** This step must be performed in the instructor's computer and consists of several smaller steps. First, the instructor must select or create a class. Second, he/she must up-load the list of students to MoCAS, either loading student by student or loading the whole class in one step (with a formatted text file). Third, the instructor must up-load the exercises to solve. Each exercise is just a piece of source code. Currently, MoCAS supports the languages Pascal, C and Java. Finally, the instructor assigns students and exercises to a class.

1. **Formation of groups of students and delivery of an exercise to each group.** MoCAS automatically generates the groups in a given class by grouping every four students. The instructor relates, by drag-and-drop, students with the group and the problem statement they will have to solve. When students log in the system with the user and password given to them by the instructor, they can only select the exercise and group they have been assigned.

2. **Collaboration within groups.** The students' view of MoCAS is different from the instructor's view. They may interact with their mobile devices or with their laptops or personal computers. In the former case, students have two tabs visible (see menu in Figures 3.a and 3.b, bottom); in the latter, they have all the information visible on the screen (see figure 4). In the following, we describe both interfaces in detail in order to obtain a more vivid impression of the tool.

(a) **(b)**

Figure 3. (a) MoCAS code tab. (b) MoCAS table tab, showing the confirm/disconfirm dialog

The device interface consists of two tabs:

a. Code tab (Figure 3.a). It contains the source code assigned to the student. Code lines are numbered to make referring to identifiers easier. Figure 3.a shows a part of an exercise that will be used along the article to illustrate MoCAS features. This exercise consists of four "blocks", namely a main program named *Prog*1 and three subprograms named *P*1, *F*1 and *P*2.
b. Table tab (Figure 3.b). It constitutes a synchronous group desktop, i.e. it allows the members of a group to communicate synchronously using their desktops. The table tab hosts a table divided into two areas (Figure 3.b, mark numbers 1 and 2). The left-hand side area displays in different rows the names of the blocks occurring in the exercise source code. Thus, Figure 3.b identifies the four blocks of our example, namely *Prog*1, *P*1, *F*1 and *P*2. The right-hand side area displays the items (i.e. identifiers) contributed by students. A student may either insert or delete his/her own items or vote the items proposed by his/her group mates. Let us review these three operations.

   Firstly, a student may insert an item by following three steps:

   1. He/she selects a block where to insert an item (Figure 3.b, mark number 1).
   2. He/she selects an identifier from a list (available in a dropdown text box with all the identifiers and the line numbers where they are declared). For example "(009)y" means that the identifier *y* is declared in line 009.
   3. He/she presses the Add button. The new identifier is inserted into the row of the selected block, meaning that this block is within the scope of the identifier.

   Secondly, a student may delete an item by clicking over the item and selecting the delete operation in the contextual menu. The student may only delete his/her contributed items.

   Finally, a student may vote the items contributed by other members of the same group as confirming or disconfirming. Voting can also be done by clicking over the item and selecting in the contextual menu the corresponding option (either to confirm or to disconfirm). The contextual menu also shows, for each item, who contributed it and the number of confirming or disconfirming votes received. This information can be used by any student to consider if deleting a contributed item (if most members of his/her group disagree with that insertion) or to reinforce subjective evidence about the correctness of the contribution.

   During inter-group discussion, the instructor may supervise the groups in the class and may even insert his/her own identifiers to promote additional discussion. The instructor may even project his/her contributions to the class for a class discussion.

Due to their greater screen size, the laptop or personal computer interfaces display (in different areas) the same information as the mobile device interface hosted in two tabs. Figure 4 shows, marked with numbers 1, 2, 3 and 4, the same contents as shown above for the code tab, the two tables and the contextual menu. Some minor enhancements are provided. In particular, items have been tagged with icons to make the identification of contributions easier (see Figure 4, mark number 3). A single person icon denotes that it is an own contribution, and a three-people icon denotes that it was contributed by another group member.

The laptop or personal computer interfaces also provide several new tabs to the student:

a. Theory tab (see Figure 4, mark number 5). The student may consult concepts about scope of identifiers in a standard format (e.g. HTML). The theory is loaded by the instructor when he/she creates an exercise.
b. Solution, group evaluation and classroom evaluation tabs. These tabs are used to show extra information to the students when the learning task is over. The solution tab (Figure 4, mark number 6) shows an automatic correction done of the class solution (explained in more detail in steps 4 and 6, as well as in Figures 5 and 6). The solution can be projected to the class. The group evaluation and classroom evaluation tabs (see Figure 4, mark numbers 7 and 8) present simple reports about the hits and misses of the student's group and of the whole class.

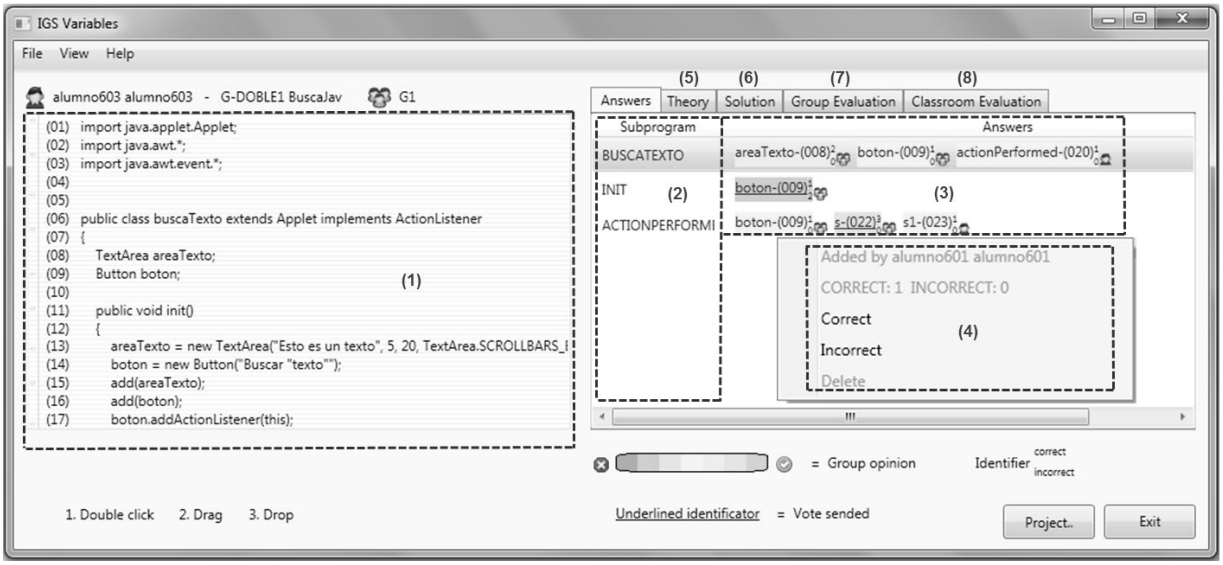

Figure 4. MoCAS interface for students in a laptop

3. **Repetition of the analysis task with another exercise**. This step is optional, at the instructor's discretion. Once all the groups have created a solution to the first given exercise, the instructor may assign a second exercise to the class just by dragging and dropping a new piece of source code. He/she asks the students to login again in MoCAS and select the new exercise. This step is similar to step 1, and step 2 must be performed again (i.e. collaborate within groups).

4. **Creation of a class solution.** When time scheduled for problem solving is over, students stop interacting with their computing devices. The instructor selects the adequate class (see Figure 5, mark number 1) and exercise (Figure 5, mark number 2), obtaining a display of the source code (Figure 5, mark number 3) and a summary of the solutions proposed by all the groups. In the column entitled "procedures" we find the program blocks (see Figure 5, mark number 4), and in the column entitled "solution" we find a summary of the solutions proposed by all the groups (see Figure 5, mark number 5). This summary constitutes a collaborative "class solution" to the exercise, which is projected to the class using MoCAS and a projector. It has the same tabular format as the solution displayed to each student for group work, composed of block names, identifiers and code lines proposed for each block. Thus, the occurrence of "(003)a" in the first row of Figure 5, means that identifier *a* declared in line 003 of the source code has *Prog*2 within its scope.

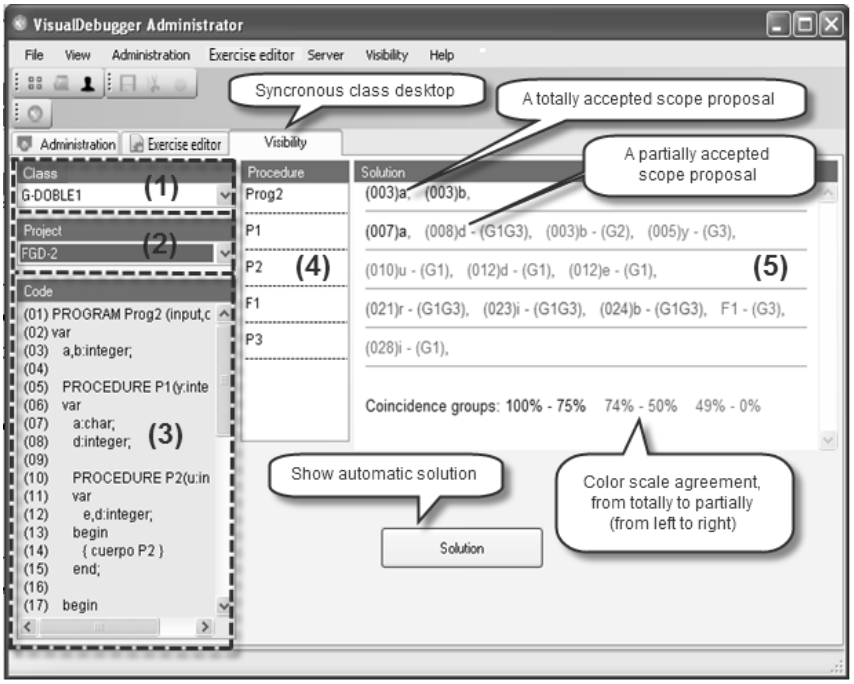

Figure 5. Class solution

The degree of global consensus about each identifier proposal is highlighted by displaying the group names that proposed it. If all the groups agree about a proposal, no group is indicated. Otherwise, at the right of the proposal, the names of the groups that suggested the identifier appear enclosed in parenthesis. For example, "(008)d-(G1G3)" means that identifier *d* declared in line 008 was only proposed by groups G1 and G3 (see Figure 5). Moreover, a coloring convention allows knowing the degree of consensus among the groups about each identifier proposal: black denotes total or almost total agreement (higher than 75%), green denotes partial agreement (50% to 74%) and red denotes low agreement (lower than 50%).

A discussion can be held in the classroom about the proposals (probably about those colored in red or in green) in a trial to make explicit misunderstandings and to obtain universal agreement. The instructor should guide and coordinate the discussion so that each group will justify their proposals to the rest of the class.

5. **Presentation of the collaborative group solutions to the whole class**. When collaborative problem solving within groups is over, every group has to present and defend their solution to the whole class by using the merged solution displayed by MoCAS. Figure 5 is shown in class by means of a projector and every group explains their contribution to the merged solution, paying special attention to the items with no total agreement.
6. **Comparison of the class solution with the automatic solution generated by MoCAS.** After the in-class discussion, the class solution is compared with the solution automatically generated by MoCAS. By clicking the solution button (see Figure 5), the automatic solution is displayed (Figure 6, mark number 2), and the class solution is updated with two pieces of additional information: missing proposals are displayed and colored in blue and wrong proposals are crossed (Figure 6, mark number 1). The instructor may then clarify any doubt about these missing or wrong proposals to fix the concepts misunderstood.

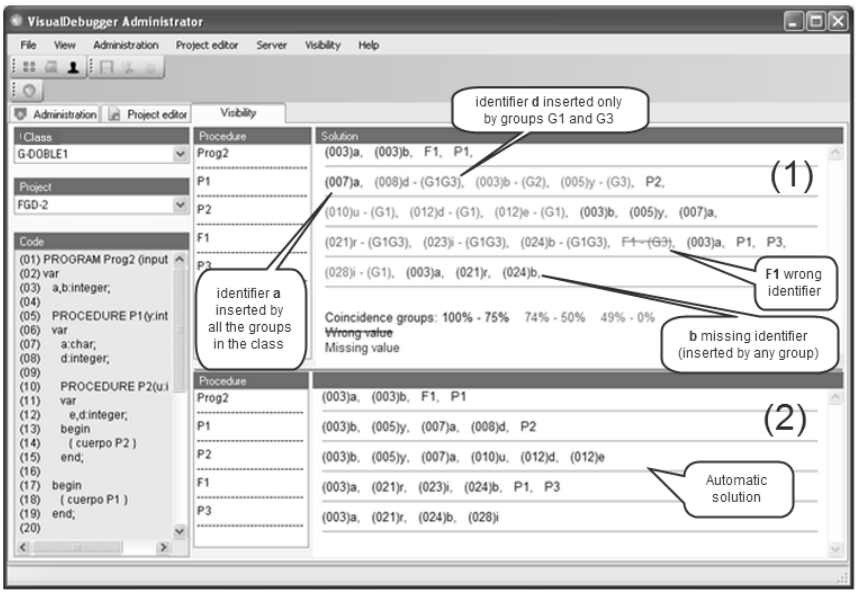

Figure 6. Automatic solution compared with the class solution

4. Evaluation

In this section, we present the evaluations conducted of the use of CIF and MoCAS for learning the concept of scope of identifiers in procedural programming languages. MoCAS was evaluated twice, the first time with respect to educational effectiveness [12] and the second one with respect to motivation [13]. In the first subsection, we present the experimental design used for both evaluations, given that most features are in common. In the following two subsections, we summarize the results obtained in their respective evaluations.

4.1. Experimental design

The participants were students enrolled in a course on introduction to programming (CS1) at our university. Four different enrolment groups were used in both evaluations: Degree in Computer Science (two groups from two campuses), Degree in Software Engineering and Degree in Computer Engineering. Each group was instructed in the topic using a different pedagogical methodology. We formed three different control groups: E1-Ind (traditional instruction), E2-Col (collaborative instruction) and E3-CIF (collaborative instruction guided by CIF). The experimental group was E4-CIF-M (collaborative instruction guided by CIF and supported by MoCAS).

The evaluation was conducted in a two-hours session. For groups E1-Ind, E2-Col and E3-CIF, the session was held at the classroom and no equipment was required, but the session for group E4-CIF-M was held at a computer laboratory. Participation was mandatory (in general, attendance to all the course activities was mandatory). However, they were informed that participation would be rewarded with 0.15 points in the final grade of the course (over a scale of 10 points).

Figure 7 summarizes the procedure followed in the two evaluations. Students had to fill in a pre-test at the beginning of the session and a post-test at the end; these tests were intended to measure students' educational gain. After the pre-test, each instructor explained scope concepts using PowerPoint and a projector. In addition, participants in the second evaluation had to fill in a motivation questionnaire at the end of the session.

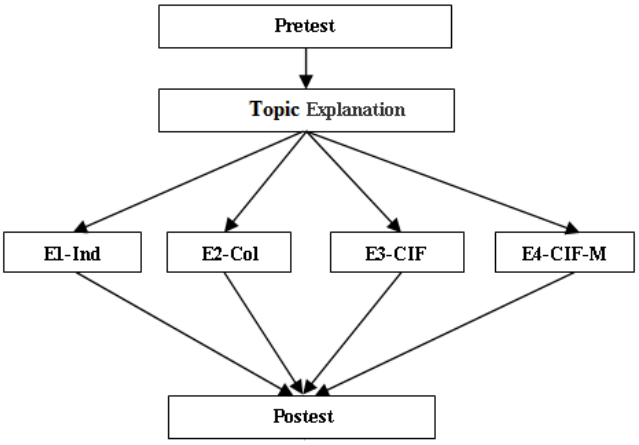

Figure 7. Evaluation procedure carried out in the four teaching approaches.

The treatment given to the different groups varied:

- E1-Ind. The class was based on a traditional lecture format. Students individually solved several exercises proposed by the instructor. At the end of the class, the instructor solved the exercises on the blackboard and discussed the different issues arisen.
- E2-Col. The class was conducted using a collaborative approach. The instructor formed groups of 4 students, being random the assignment of students to groups. Each group solved collaboratively several exercises. Finally, a voluntary group wrote their solutions on the blackboard, and the rest of students and the instructor discussed their correctness.
- E3-CIF. The class was instructed using the CIF methodology. The group formation phase was similar to E2-Col. Only one statement was delivered due to the time restriction of two hours. The instructor delivered the problem statement on paper and projected the source code to the class. Each group worked independently and built a scope table. Some groups wrote their respective scope table on the blackboard, presented it to the class and argued about its correctness. Students and the instructor discussed those proposals. Finally, the instructor explained the mistakes committed and made comments on the different issues that had arisen.
- E4-CIF-M. Group formation was supported by MoCAS and students started a MoCAS session using their mobile devices (in the educational effectiveness experiment) or their PCs (in the motivation experiment). Only one statement was delivered due to the time restriction of two hours. The instructor demonstrated how to use MoCAS over the projector and explained the problem statement. Then, students in each group used MoCAS to make their proposals of scope table, and to argue and achieve consensus about it. When students had collaboratively elaborated their groups' scope table, the tables were projected by MoCAS on the blackboard through the projector. The automatic solution generated by MoCAS was also displayed, including the highlighting of different parts. A member of each group presented and defended their proposal to the class. The instructor and the students discussed those proposals. Finally, the instructor explained with the assistance of MoCAS the mistakes committed and made comments on the different issues that had arisen.

4.2. Evaluation of learning efficiency

This evaluation was held in the academic year 2010-11. The population was formed by 215 students. The dependent variable "learning efficiency" was intended to measure the level of knowledge acquired by a student on the topic under study, and was defined as the difference between his/her score in the post-test and in the pre-test. The pre-test and the post-test were made out of multiple-choice questions with three possible answers each one. The pre-test consisted of 12 questions, and the post-test consisted of 11 questions. The questions in both tests were different.

The learning efficiency mean measured for the four groups (E1-Ind, E2-Col, E3-CIF and E4-CIF-M) was 1.24, 2.04, 3.36 and 4.29, respectively [12]. Therefore, the third and fourth groups, both based on the CIF instructional framework (E3-CIF and E4-CIF-M), obtained the highest knowledge increases. We determined whether these

results were statistically significant with a p-value=0.0083 (due to the correction of Bonferroni, p-value=0.05/6):

- E3-CIF obtained higher grades than E1-Ind, not being statistically significant but close to (p=0.015).
- E4-CIF-M obtained higher grades than E1-Ind and E2-Col, being these results statistically significant (p=0.003 and p=0.000, respectively).

In summary, we may claim higher learning efficiency for the joint use of CIF and MoCAS with respect to other instructional methods.

4.3. Evaluation of motivation

This evaluation was held in the academic year 2011-12. The population was formed by 139 students. Students were given an EMSI motivation questionnaire to fill in at the end of the session. This questionnaire is based on self-determination theory [13], [15]. There are several dimensions of motivation depending on the level of self-determination, ranging through a continuum from higher to lower self-determination:

1. Intrinsic motivation refers to doing something because it is inherently interesting or enjoyable.
2. Extrinsic motivation via identified regulation occurs when the behavior is considered important for the subject's goals and values.
3. Extrinsic motivation via external regulation refers to doing something because it leads to a separable outcome, e.g. to obtain a reward or to avoid a punishment.
4. Amotivation occurs when individuals do not perceive the contingencies between the behavior and its consequences, and behavior has not intrinsic or extrinsic motivators.

The motivation mean measured for the four groups (E1-Ind, E2-Col, E3-CIF and E4-CIF-M) was 4.11, 3.70, 4.15 and 4.94, respectively [13]. Therefore, students in E1-Ind exhibited the lowest motivation while students in E4-CIF-M exhibited the highest. We determined whether these results were statistically significant and we obtained that students instructed with E4-CIF-M were statistically more motivated than those instructed with E2-Col ($p<0.0083$).

We also wanted to know the results with respect to the four dimensions of motivation, obtaining the following statistically significant results:

- Students instructed with E3-CIF and especially E4-CIF-M exhibited higher intrinsic motivation than the other two groups.
- Students instructed with E4-CIF-M exhibited higher extrinsic motivation via identified regulation than the other groups.
- Students instructed with E1-Ind exhibited higher extrinsic motivation via external regulation that students instructed collaboratively without the use of CIF, E2-Col.
- Students instructed with E4-CIF-M exhibited higher amotivation than groups E2-Col and E3-CIF.

In summary, our results suggest that the joint use of CIF and MoCAS are associated with higher levels of intrinsic and extrinsic motivation. This positive result may contribute to higher performance of students. However, it is also, unexpectedly, associated with amotivation. We felt unable to give an explanation without having available more founded data, thus it is an open issue for the future.

5. Related work

There are a myriad of systems designed to enhance the learning of programming [6]. In the particular domain of scope of identifiers, we remark the problet developed by Amruth Kumar [16]. According to Kumar, a problet is a tutor system that combines problem generation, automatic correction, grading and feedback to the student; restricted forms of visualization also are common. In the cited article, Kumar presents a tutor aimed at learning scope of identifiers, describes the design decisions adopted and presents an evaluation of educational effectiveness, where students obtained a statistically significant enhancement in their performance. Nevertheless, problets is not a collaborative tool.

To the best of our knowledge, there are no other tools specifically designed for learning scope of identifiers. There are some experiences in applying to the programming domain CSCL systems designed for other domains (e.g. DOMOSIM-TPC [17]). Other experiences report on the use of general-purpose CSCL systems in programming

engineering courses (CollaborativeWeb [18]). Nevertheless, their use poses extra effort to programming instructors. Therefore, it is more adequate to compare MoCAS with systems developed for programming education.

We may classify programming learning CSCL tools into two categories: general CSCL programming learning tools and CSCL programming learning tools aimed at learning programming concepts. General programming learning tools are used by students to write source code. We can highlight two general programming learning tools: SICODE and COLE-Programming. SICODE [19] is a web collaborative system for learning Java which is focused on detecting, classifying and monitoring compilation errors. The system supports collaborative edition of source code files using the CVS version control tool, and provides a tool for arguing and discussing among students by using text messages. COLE-Programming [20] is an Eclipse plug-in derived from COALA [21] which integrates collaborative tools (i.e. a chat, a forum and a voting system). The tool is a distributed system aimed at algorithm courses, where students may write source code and the instructor may write annotations on it. The chat and the forum can be used by students to argue and discuss collaboratively and they allow students to publish compilation errors.

A second type of tools comprises CSCL tools for learning programming concepts. We may highlight three tools: JeCo, SICAS-COL and HabiPro. JeCo [22] integrates two previously existing tools: the program animation tool Jeliot 3 for Java coding and the collaborative story telling tool Woven Stories. JeCo recognizes when a section of text contains Java code that students can animate by clicking over it using the right-hand side mouse button. Students can simultaneously watch the same visualization and discuss about it. SICAS-COL [23] is a tool resulting from integrating SICAS and PlanEdit. It is organized into three workspaces: individual work, group discussion and sharing of results. When a student is in his/her individual workspace, he/she can create, modify or visualize solutions. In the discussion space, students can discuss a solution and propose alternatives. The results workspace is used to store solutions proposed and other documents related to the given problem. HabiPro [24] is an adaptive system intended to assist students in developing programming skills like observing, reflecting and organizing. HabiPro supports collaborative discussion using a chat and it provides awareness through list of users (it shows the name and a photo of users).

If we compare MoCAS with these tools, we come to the following conclusions:

1. There is only one additional tool (i.e. a problet) aimed at the same domain (scope of identifiers) as MoCAS. However, it is not a collaborative tool.
2. Other CSCL tools were developed for PCs, making face-to-face interaction difficult. Besides, they are mostly centered on visualizations, not covering other specific programming learning aspects.
3. Other CSCL tools are not a part of an instructional framework, thus it depends on the instructor' skills to take advantage of them.
4. MoCAS is the only tool developed for both desktop PCs and mobile devices.
5. MoCAS is the only tool that has been used to evaluate students' educational efficiency and motivation.

6. Discussion

We may highlight two issues of our work in relation to software engineering practices, namely the design and evaluation of MoCAS. Requirements gathering, specification and design are key steps in the construction of software tools. General methodologies used for these tasks require knowledge of the target domain. Bloom's taxonomy provides the knowledge of and terminology about the target domain of education. Actually, Bloom's taxonomy is a framework commonly used in computing education [25]. It has also been used as a framework to analyze and design educational software aimed at specific levels of Bloom's taxonomy [26], e.g. GreedEx is an experimentation tool aimed at the comprehension, analysis and evaluation levels [27]. Consequently, the CIF framework gives strong assistance to the designer of CSCL systems aimed at the analysis level of Bloom's taxonomy. He/she must instantiate GCs into corresponding DCs for the target domain and the resulting DCs can be used as partial specifications of the intended tools. This design approach was actually used to design and construct MoCAS.

CIF has similarities to the activity theory, a framework often used for user interface design and construction [28], in particular CSCL systems [29]. We do not have space here to present activity theory in detail, but we may briefly introduce some key concepts. The theory distinguishes three hierarchical levels of abstraction, called activity, action and operation. An activity is a part of our dairy behavior and it is guided by a motive. An activity is composed of a

set of steps, called actions, which can be characterized by goals. Finally, an action is accomplished by a set of operations, which can be performed if certain conditions are satisfied.

It is straightforward to notice some resemblance between CIF and the activity theory. A CIF generic (or concrete) card defines a "task" and decomposes it into a sequence of "activities", each one accomplished by a sequence of "atomic actions". However, we must be careful because terminology can be misleading in a comparison between the activity theory and CIF. A CIF task corresponds to an action in the activity theory because an educational goal is stated. CIF activities correspond to actions of finer granularity. Finally, an atomic action is usually an operation and, in some cases, a very specific, domain-dependent activity. A consequence of this hierarchy is that we were able to elaborate a list of atomic actions common in collaborative settings, thus providing a shared language to developers and instructors. Furthermore, most atomic actions are close to operations in an interactive system. Consequently, the design of the actual interactive functions supported by the CSCL system is easier, as was the case for MoCAS development.

As stated in the first paragraph, a second issue of interest to the software engineering community regarding MoCAS is its evaluation. There are different factors that can be evaluated in a software product, e.g. conformance to the user needs or usability. Usability always is an important concern and it should never be neglected. We conducted usability evaluations by experts [30], i.e. by instructors. In addition, personal interviews were carried out with students about their personal perception regarding the use of MoCAS. The answers of both kinds of users guided us through a number of MoCAS prototypes where we were able to add extra GUI functionality and to improve the user interface of platforms supported by MoCAS. However, the most important issue for an educational tool is educational effectiveness. Once usability was assured, we were in a better position to evaluate its educational effectiveness [31]. As we showed above, we obtained statistically significant learning improvements in students using MoCAS (and therefore CIF) with respect to students not using either CIF or MoCAS.

User satisfaction is an important issue in the development and design of interactive systems. Its instantiation in educational settings is students' motivation. Thus, we also conducted an evaluation of students' motivation regarding the proposed activity. We obtained statistically significant motivation improvements in students using MoCAS (and therefore CIF) with respect to students receiving collaborative instruction but non-based on CIF. The results obtained here are more complex to interpret. Students using MoCAS exhibited higher intrinsic and extrinsic motivation via identified regulation than other students. However, we also measured higher amotivation in students using MoCAS than the other groups being instructed collaboratively without using MoCAS. These mixed results deserve deeper studies in order to be able to explain them and act accordingly.

7. Conclusion

We have given a comprehensive presentation of a mobile collaborative tool called MoCAS aimed at supporting the instruction of scope of identifiers. We may highlight three contributions. Firstly, we present a novel collaborative instructional framework, called CIF, intended to assist in specifying collaborative activities. CIF is aimed at the analysis level of Bloom's taxonomy because this level is especially adequate for collaborative learning. Secondly, the specification and development of MoCAS were explicitly driven by pedagogical goals and atomic actions declared in the CIF instructional framework. MoCAS is a multimodal tool developed for mobile devices, laptops and personal computers, thus facilitates face-to-face interaction among students. Furthermore, few systems are aimed at the domain of scope of identifiers in procedural programming languages. Thirdly, we present two evaluations of CIF and MoCAS, resulting in higher educational efficiency and higher motivation. Note that both evaluations compared four instructional approaches, which is an uncommon procedure for these evaluations.

Based on the results obtained and our previous experience with other tools, we may conclude that the use of explicit pedagogical goals and a clear instructional approach to the design of learning tools seems to constitute a solid approach to technology development in educational contexts. The main drawback of MoCAS, however, is its tight link to a narrow domain, namely scope of identifiers in programming languages. However, most of its functions can be stated in terms independent from the domain. In a trial to remedy this drawback, we have analyzed the atomic actions contained in DC1 and we have identified all the functions of MoCAS that are domain-independent. A CSCL domain-independent tool aimed at supporting CS1 is under development. An open challenge will be the adoption by instructors of such a generic, analysis-oriented CSCL tool.

Acknowledgments

This work was supported by the Ministry of Economy and Competitiveness under research grant TIN2011-29542-

C02-01, the Region of Madrid under research grant S2013/ICE-2715, and the Universidad Rey Juan Carlos under research grant 30VCPIGI05.

References

[1] M. Riojas, S. Lysecky and J. Rozenblit, Educational technologies for precollege engineering education, *IEEE Transactions on Learning Technologies*, **5**(1), 2012, pp. 20-37

[2] ACM and IEEE-CS, The Joint Task Force on Computing Curricula, *Computer Engineering 2004 - Curriculum guidelines for undergraduate degree programs in computer engineering*, ACM Press, 2004

[3] ACM and IEEE-CS, The Joint Task Force on Computing Curricula, *Software Engineering 2004 - Curriculum guidelines for undergraduate degree programs in software engineering*, ACM Press, 2004

[4] S. Fincher and M. Petre, *Computer Science Education Research*, Routledge, 2004

[5] A. Robins, J. Rountree and N. Rountree, Learning and teaching programming: A review and discussion, *Computer Science Education*, **13**(2), 2003, pp. 137-172

[6] C. Kelleher and R. Pausch, Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers, *ACM Computing Surveys*, **37**(2), 2005, pp. 83-137

[7] M. Prince, Does active learning work? A review of the research, *Journal of Engineering Education*, **93**(3), 2004, pp. 223-231

[8] F. Dochy, M. Segers, P. Van den Bossche and D. Gijbels, Effects of problem-based learning: A meta-analysis, *Learning and Instruction*, 13(5), 2003, pp. 533-568

[9] T. D. Koschmann, *CSCL: Theory and Practice of an Emerging Paradigm*, Routledge, 1996

[10] B. S. Bloom, M. Engelhart, E. J. Furst, W. H. Hill and D. R. Krathwohl, *Taxonomy of Educational Objectives: Handbook I: The Cognitive Domain*, Addison-Wesley, 1959

[11] D. Boud, R. Cohen and J. Sampson. Peer learning and assessment, *Assessment & Evaluation in Higher Education*, **24**(4), 1999, 413-426

[12] L. M. Serrano-Cámara, M. Paredes-Velasco and J. Á. Velázquez-Iturbide, Evaluation of a collaborative instructional framework for programming learning, *Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education*, 2012, 162-167

[13] L. M. Serrano-Cámara, M. Paredes-Velasco, C.-M. Alcover and J. Á. Velazquez-Iturbide, An evaluation of students' motivation in computer-supported collaborative learning of programming concepts, *Computers in Human Behavior*, **31**, 2014, pp. 499-508

[14] R. M. Ryan and E. L. Deci, Self-determination theory and the facilitation of intrinsic motivation, social development, and well-being, *American Psychologist*, **55**(1), 2000, pp. 68-78

[15] R. M. Ryan and E. L. Deci, Intrinsic and extrinsic motivations: Classic definitions and new directions, *Contemporary Educational Psychology*, **25**(1), 2000, pp. 54-67

[16] A. N. Kumar, Generation of problems, answers, grade, and feedback - Case study of a fully automated tutor, *ACM Journal on Educational Resources in Computing*, **5**(3), 2005, article 3

[17] M. Á. Redondo and C. Bravo, DOMOSIM-TPC: Collaborative problem solving to support the learning of domotical design, *Computer Applications in Engineering Education*, **14**(1), 2006, pp. 9-19

[18] D. A. Fuller and A. F. Moreno, Experimenting with a computer-mediated collaborative interaction model to support engineering courses, *Computer Applications in Engineering Education*, **12**(3), 2004, pp. 175-188

[19] J. Pérez, M. del Puerto Paule Ruiz and J. Lovelle, SICODE: A collaborative tool for learning of software development, *Proceedings of the IV International Conference on Multimedia and Information & Communication Technologies in Education*, 2006

[20] F. Jurado, A. I. Molina, M. A. Redondo and M. O. Cantero, COLE-Programming: Incorporando soporte al aprendizaje colaborativo en Eclipse, *IEEE Revista Iberoamericana de Tecnologías del Aprendizaje*, **7**(3), 2012, pp. 121-130

[21] F. Jurado, A. I. Molina, M. A. Redondo, M. Ortega, A. Giemza, L. Bollen and H. U. Hoppe, Learning to program with COALA, a distributed computer assisted environment, *Journal of Universal Computer Science*, **15**(7), 2009, pp. 1,472-1,485

[22] A. Moreno, N. Myller and E. Sutinen, JeCo, a collaborative learning tool for programming, *Proceedings of the IEEE Symposium on Visual Languages and Human Centric Computing*, 2004, pp. 261-263

[23] B. Rebelo, A. Mendes, M. Marcelino and M. Redondo, Sistema colaborativo de suporte à aprendizagem em grupo da programação – SICAS-COL, *Proceedings of the VII International Symposium of Computers in Education*, 2005, pp. 113-117

[24] A. Vizcaíno, J. Contreras, J. Favela and M. Prieto, An adaptive, collaborative environment to develop good habits in programming, *Intelligent Tutoring Systems*, Springer, 2000, pp. 262-271

[25] U. Fuller, C. G. Johnson, T. Ahoniemi, D. Cukierman, I. Hernán-Losada, J. Jackova, E. Lahtinen, T. L. Lewis, D. McGee Thompson, C. Riedesel and E. Thompson, Developing a computer science-specific learning taxonomy, *ACM SIGCSE Bulletin*, **39**(4), 2007, pp. 152-170
[26] I. Hernán-Losada, C. A. Lázaro-Carrascosa and J. Á. Velázquez-Iturbide, On the use of Bloom's taxonomy as a basis to design educational software on programming, *Proceedings of the 2004 World Conference on Engineering and Technology Education*, 2004, pp. 351-355
[27] J. Á. Velázquez-Iturbide, O. Debdi, O., N. Esteban-Sánchez and C Pizarro, GreedEx: A visualization tool for experimentation and discovery learning of greedy algorithms. *IEEE Transactions on Learning Technologies*, **6**(2), 2013, pp. 130-143.
[28] V. Kaptelinin and B. A. Nardi, *Acting with Technology – Activity Theory and Interaction Design*, The MIT Press, 2006
[29] B. Collis and A. Margaryan, Applying activity theory to computer-supported collaborative learning and work-based activities in corporate settings, *Educational Technology Research & Development*, **52**(4), 2004, pp. 38-52
[30] J. Nielsen and R.L. Mack (eds.), *Usability Inspection Methods*, John Wiley, 1994
[31] J. Á. Velázquez-Iturbide, A. Pérez-Carrasco and O. Debdi, Experiences in usability evaluation of educational programming tools, *Student Usability in Educational Software and Games: Improving Experiences*, Carina González (ed.), IGI Global, 2013, pp. 241-260

Biographies

Luis Miguel Serrano-Cámara obtained the degree of Technical Engineering in Computer Science by the University Complutense of Madrid in 1997. In 2007 and 2010 he studied both a Masters in Information Technology and Hardware and Software Systems, in the Rey Juan Carlos University. At the present time, he is performing his PhD on the field of collaborative learning in education, focusing on its effects on learning efficiency and motivation. From 1997 to 2004 he worked as software developer and development team leader. Since 2004 he has taught Vocational Training in Computers at the Secondary School Enrique Tierno Galván (Parla), combining both his teaching in high school and being a part-time associate professor at the University Rey Juan Carlos in Madrid..

Maximiliano Paredes-Velasco received the Computer Science degree from Universidad de Sevilla (Spain) in 1998 and Ph.D. degree in Computer Science from Universidad de Castilla – La Mancha (Spain) in 2006. In 1998 he joined the Universidad de Alcalá de Henares, where he was an Assistant Professor. In 1999 he joined Universidad Rey Juan Carlos, where he is an Assistant Professor and he is a member of the Departamento Ciencias de la Computación, Arquitectura de Computadores, Lenguajes y Sistemas Informáticos y Estadística e Investigación Operativa. At the present time, he is Head of infrastructure and laboratories at Universidad Rey Juan Carlos. His research is focused on computer supported collaborative learning, ubiquitous computing and computer human interaction.

J. Ángel Velázquez-Iturbide received the Computer Science degree and the Ph.D. degree in Computer Science from the Universidad Politécnica de Madrid, Spain, in 1985 and 1990, respectively. In 1985 he joined the Facultad de Informática, Universidad Politécnica de Madrid. In 1997 he joined the Universidad Rey Juan Carlos, where he is currently a Professor, as well as the leader of the Laboratory of Information Technologies in Education (LITE) research group. His research areas are software and educational innovation for programming education, and software visualization. Prof. Velázquez is an affiliate member of IEEE Computer Society and IEEE Education Society, and a member of ACM and ACM SIGCSE. He is the Chair of the Spanish Association for the Advancement of Computers in Education (ADIE).