This document is published in:

Computers & Education, (2013), 61, 33-42.
DOI: http://dx.doi.org/10.1016/j.compedu.2012.09.004

Addressing drop-out and sustained effort issues with large practical groups using an automated delivery and assessment system

Luis de-la-Fuente-Valentín, Abelardo Pardo, Carlos Delgado Kloos

Department of Telematics Engineering, University Carlos III of Madrid, Av. Universidad, 30, 28911 Leganés (Madrid), Spain

E-mail addresses: lfuente@it.uc3m.es (L. de-la-Fuente-Valentín), abel@it.uc3m.es (A. Pardo), cdk@it.uc3m.es (C. Delgado Kloos).

Abstract: The acquisition of programming skills specially in introductory programming courses poses an important challenge for freshmen students of engineering programs. These courses require students to devote a sustained effort during the whole course and a failure to do so may contribute to not passing the course. However, it is difficult for the teaching staff to deploy measures to enforce a pattern of continuous work without significantly increasing the required management tasks. A significant reduction of this workload can be achieved with the automation of time consuming tasks such as the delivery of activities or submission grading. This paper presents a case of study where a technology based orchestration of learning scripts was applied in a large enrollment course to promote student sustained effort through the course and keep the workload on teaching staff within reasonable margins. The orchestration system, based on IMS Learning Design and Generic Service Integration, automatically evaluated the student submissions and gradually unlocked the following activities depending on the received results. Such system was used during a semester supporting 425 students and 8 instructors. The analysis of the case of study followed a mixed method based on qualitative and quantitative data, and revealed a significant reduction of the orchestration workload on the teaching staff, allowing the strategy of continuous work to be applied in a course with high enrollment. Additionally, the application of these techniques show statistically significant differences in the drop out rate with respect to previous editions of the course.

Keywords: Project based learning, Programming, IMS Learning Design, Automatic grading, Sustained effort.

1. Introduction

Programming courses are an important part of engineering curricula and an essential part of most engineering programs (ACM/IEEE, 2005). However, learning to write computer programs is recognized to be a challenging subject for students due to the required logical thinking and the implied abstract concepts (Milne & Rowe, 2002; Robins et al., 2003; White & Sivitanides, 2002). Programming also poses a challenge to instructors because of the difficulty of grading student code in a fair and time-efficient manner (Cheang, 2003; Higgins, Gray, Symeonidis, & Tsintsifas, 2005).

As a consequence of these difficulties, programming courses suffer from high drop out rates, especially in the case of freshman students as documented by McKinney and Denton (2004). The more general problem of high attrition rate in generic engineering courses has been addressed in engineering education research, for example in (Brusilovsky, Kouchnirenko, Miller, & Tomek, 1994; Fincher, 1999). Poor student performance is, among other factors, caused by the lack of time and motivation (Kinnunen & Malmi, 2006). Moreover, programming is completely new for most freshman students and they find the required concepts difficult to understand (Milne & Rowe, 2002). This fact affects their perceived self-efficacy, which is another relevant factor that is reported to cause the observed high drop out rates (Wiedenbeck, 2005).

In order to increase student motivation, and therefore engage them in the course, a widely used technique is the application of Project Based Learning (PBL). Apart from the motivational gains, PBL promotes active learning, collaborative skills and the acquisition of problem solving abilities (Yadav, Subedi, Lundeberg, & Bunting, 2011) by the introduction of "a complex task created by the need to design, create, build, repair, and/or improve something" (Burgess, 2004). Since it appeared in the 1950s, PBL was initially used in medical education and

now is being used in other educational areas including engineering education and, more specifically, computer programming courses (Robins, Rountree, & Rountree, 2003).

Despite all the benefits reported when using problem solving techniques, the cost of such methodology significantly hinders its adoption by practitioners (Martinez-Mones et al., 2005). Small-group methodology is expensive in terms of instructor time and also exposes instructors to additional stress (Berkson, 1993). In PBL, the instructor becomes a learning facilitator and must pay individual attention to all student groups. As a conclusion, although beneficial, PBL is difficult to apply in high enrollment courses (Albanese & Mitchell, 1993).

A workload reduction can be achieved through the automation of several time consuming tasks with a technology based orchestration for the delivery of learning scripts. With such an approach, a set of pre-programmed activities are presented to the students when they finish a task or accomplish a certain goal. This paper presents a case of study focused on the use of an orchestration system aimed at the automatic delivery of learning scripts for the reduction of the workload imposed by PBL strategies. The proposed system uses IMS Learning Design (IMS LD) (IMS Global Learning Consortium, 2003) as the formalism to deliver the activity flow, while case-specific tasks are integrated in the course flow through Generic Service Integration (GSI) (de-la-Fuente-Valentín, Pardo, & Kloos, 2011).

The case of study was carried out during the 2009/2010 edition of a freshmen programming course with 425 students at a higher education institution. During the course, students worked in pairs to develop the solution of a complex problem. The experimental setting was conceived to evaluate the effect of the orchestration system in the following aspects of the course: teacher workload, drop out rate, overall course results, and students engagement. The results show that the new method achieved significant reduction in teaching staff workload, a statistically significant reduction in attrition rate when compared to previous editions of the same course, and was well received by students.

The rest of the paper is organized as follows: Section 2 presents a review of the literature on learning scripts applied to computer programming courses. Then, Section 3 describes the experimental setting, the orchestration system under study and the enacted learning script. After that, Section 4 explains the research methodology used to validate the proposal and how the required data was gathered. Evaluation results are presented and discussed in Section 5. Finally, Section 6 draws the conclusions of the presented work.

2. Literature review

The orchestration system used in this paper combines several ideas in the same educational setting. First, the use of learning scripts to achieve effective orchestration of learning activities in a large group. Second, project-based learning to promote a continuous work and engage students in course activities. And third, the use of case-specific tools to support specific needs in learning activities. In our experimental setting, the case-specific tool was a test-case based tool for semiautomatic grading of source code. We have not found in the literature any previous work that combines these three elements.

2.1. Learning scripts

Project-based learning (PBL) techniques have been shown effective to increase student motivation and reduce drop out rates (Albanese & Mitchell, 1993). With PBL, learning comes from the process of understanding and solving an open-ended problem in a collaborative manner. This method helps students understand the whole problem. It is widely used in modern education. There is also extensive research literature that describes the use of PBL techniques in the context of programming courses (dos Santos, da Conceição Moraes Batista, Cavalcanti, Albuquerque, & Meira, 2009; Tseng, Chiang, & Hsu, 2008; Zhang & Du, 2008).

Nuutila, Törmä, Kinnunen, and Malmi (2008) apply this methodology with a significant reduction in attrition rate. However, when they explored tutor-less scenarios, they claimed that the process is difficult to be scaled to large groups due to the lack of tutoring and classroom resources. Köse (2010) proposes to support PBL task assignment through a web based platform were both instructors and students are guided by a task-flow manager. The automated flow management reduced instructor workload and increased the scalability of the methodology, but was only applicable in the described context because of the case specific functionalities of the platform.

The automated flow management is similar to the idea of *learning scripts*. The term "script", when applied in a pedagogical context, refers to a method to structure a learning process (Hernández-Leo, Asensio-Pérez, & Dimitriadis, 2005). In a learning script, course participants are guided through the flow of activities previously defined by the authors of the script. The approach is used in collaborative learning, where the rationale is to structure collaborative learning processes in order to trigger group interactions that may be rare in free collaboration (Dillenbourg & Tchounikine, 2007; Stahl, 2005). Several studies show the effectiveness of scripts to support collaborative learning in certain educational scenarios. For instance, the *Universante* script has been used in the health education community for medical students from different countries (Berger et al., 2001; Dillenbourg & Jermann, 2007). Another example is the *ArgueGraph* script, which is structured as a set of activities that aims "to make students understand the relationship between learning theories and the design choices in courseware development" (Jermann & Dillenbourg, 2003).

The use of learning scripts in the support of PBL activities allows different scenarios to be supported within the same technological setting. One example of such use is the work presented by Garcia-Robles, Diaz-del Rio, and Vicente-Diaz (2009), that proposes a method to handle a PBL activity flow with the IMS LD specification (IMS Global Learning Consortium, 2003). This method focuses also on web usage and enhances reusability and interoperability of the designs. However they used the specification features as a design procedure, but not as a delivery and enactment method.

The IMS Learning Design (IMS LD) specification is considered the de-facto standard for course modeling with learning scripts (Martinez-Ortiz, Sierra, & Fernandez-Manjon, 2009). The data model provided by the specification allows to capture an arbitrary learning flow (i.e. a sequence of activities ordered according a certain pedagogical model) into a single-file package called Unit of Learning (UoL). Such packages can be loaded in one of the existing run time engines (aka. orchestration systems) and the learning activities are then delivered to the course participants according to the instructions contained in the UoL. IMS LD uses the concept of "user role", so that each role in a course has a different relationship with the content and with the learning flow. In practice, it means that each participant may receive different learning material depending on the role that he or she is playing. Such capacity allows IMS LD to be used for the orchestration of collaborative learning scripts (Hernández-Leo et al., 2007; Valdivia, Nussbaum, & Ochoa, 2009). Moreover, the definition of properties and

conditions makes possible the implementation of adaptive learning material, as pointed out in (Towle & Halm, 2005, Ch. 12) and (Burgos, Tattersall, & Koper, 2007): course participants may follow different learning paths, or work at different paces with the same course material.

One of the drawbacks of the IMS LD specification is the lack of a method to integrate the use of external web tools in the course flow. As a consequence, neither collaboration nor adaptation scenarios can be described with IMS LD. Generic Service Integration (GSI) was proposed to relax this restriction by allowing course designers to include the use of external web based tools and define an orchestration that may react to information produced in these tool (de-la-Fuente-Valentín et al., 2011).

The work presented in this paper extends these ideas by providing a system based on IMS Learning Design and Generic Service Integration to deploy PBL activities in large enrollment classes while maintaining teaching staff workload to a minimum.

2.2. Automatic program grading

Grading student programming assignments manually has a number of recognized drawbacks. Kurnia et al. (2001) analyses the issue from a theoretical perspective and identifies problems such as the required time, the difficulty and inconsistency of re-grading or the lack of defense against malicious programs, just to mention some. A similar analysis is performed by Cheang (2003), who identifies *correctness*, *robustness*, *efficiency* and *maintainability* as the main components involved in this type of grading.

The use of test cases to improve the grading process are frequently used as reported in the scientific literature. One proposed approach is based in the test-first paradigm, where students develop their own test cases (Edwards, 2003). In other examples, student submissions are automatically checked by the test cases developed by instructors. Such test cases are usually only visible to the teaching staff, and the student view is limited to the feedback offered by the system (Arnow & Barshay, 1999; Feldman & Jones, 1997; Jackson & Usher, 1997; Jones, 2000; Reek, 1996). A half-way solution is illustrated by the *marmoset* platform, a testing system that defines different types of test cases: student test, public test, release tests and secret tests (Spacco et al., 2006). A common claim of these studies is that managing a large number of student assignments is only possible through the automation of the grading process.

The fairness of automatic grade systems is analyzed in detailed by Cheang (2003), concluding that test cases are helpful to evaluate the correctness, robustness and efficiency, but are not suitable to determine the maintainability of the code. Thus, a human review must always be present in the process. Another drawback of code testing as a grading method is the need to reduce the assignment size due to the difficulty to create test cases for large programs, which is typically the case of PBL assignments.

3. Description of the case of study

3.1. Course context

The case of study presented in this document is a second semester programming course in a higher education institution. The course is the continuation of a first semester introduction to programming. Students that failed the first semester course typically abandon the second semester course during the initial weeks. The course is part of four different programs all of them with a telecommunications engineering background. The number of students is consequently large, and so is the number of tutors. The course grading policy is as follows:

a) 5 5-min tests performed during lectures (5%)
b) 2 midterm exams (20%)
c) Software projects, based on Project Based Learning methodology (20%)
d) The cooperation and work shown during the laboratory sessions (10%)
e) A final practical exam at the laboratory (15%)
f) A final paper-based exam (30%)

Student time dedication corresponds to 6 ECTS (European Credit Transfer System) credits, equivalent to 150–180 h of student work. Ideally, students should devote to each of the above items as much time as they weight in the overall evaluation system. That is, the projects should represent 30–36 h of dedication.

The orchestration system described in this section was used for the scaffolding of the two software projects (item c. in the list). A discussion of the complete course grading methodology is out of scope of this document, and it is mentioned here solely to contextualize the experience.

In previous editions of the course, students worked in pairs (also referred as “teams”) in the software projects, and they performed the tasks in an unsupervised environment, with the tutors available for help, and making a single project submission at the end of the project period. Although the scheme was well considered by instructors and students, it had several flaws:

- Due to the single submission at the end of the project period, instead of working at a sustained pace, students started the tasks a few days before the deadline.
- Several students dropped the course due to the peak in course work.
- Several teams did not understand the tasks and waited too long to ask the instructors.
- Several students simply misunderstood the required tasks.

The submitted code was, in most cases, below the expectations. There were many submissions that did not fulfill the requirements, and even code that did not compile correctly (which was considered as a minimum requirement).

The proposed solution to tackle all these flaws was to promote sustained work and better student engagement by producing a scripted version of the project during the whole project phase. This new orchestration should reduce the drop out rate, increase the quality of the submitted code, but maintaining instructor workload within reasonable margins. A detailed description of the architecture supporting the enactment of the learning script is presented in Section 3.3.

3.2. The scripted project

The project scripting with automatic orchestration was the only modification introduced in the 2009/2010 edition of the computer programming course. A total of 425 students enrolled in the course. Students were divided into 12 groups ranging from 21 to 46 members, and each group was assigned an instructor, for a total of 8 instructors. Group schedules were all coordinated with minor variations due to calendar restrictions. In other words, all students performed the same tasks, but with slight variations in the dates.

At the start of the project, the students were given an overview of the application to be developed and were also provided with the instructions to submit their solutions. This information was accompanying the first programming assignment. When a new solution was received, the test cases previously created by the teaching staff were automatically applied and a feedback report with the results was sent back to the students. The students were allowed to access the next assignment only when their submissions passed the test cases. With this approach, the overall methodology was adapted to the pace of each team while maintaining a common sequence of tasks.

To encourage a sustained effort, a deadline was set for each submission. Teams that submitted a successful solution before the deadline were granted access to the next task immediately, so that they had extra time to work on it. Those teams that did not submit a successful solution before the expected date could still keep working in the task, but they were not allowed to submit the last part of the project. That is, penalized teams were not allowed to finish the project entirely, but they were still allowed to develop the rest of the tasks in the expected order. The activity flow for the entire experience is depicted in Fig. 1.

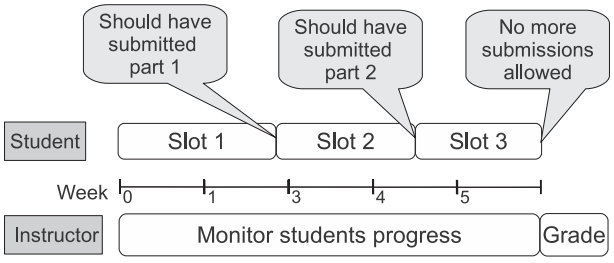

Fig. 1. Activity flow for students and instructors.

Students were supposed to complete the activities outside the classroom with unsupervised work. Additionally, there were two laboratory sessions dedicated to work in the project, one in the first week and the other in the middle of the development process. In these sessions, instructors explained the software requirements for the application and solved student doubts. Aside from these two sessions, students could visit the instructor during office hours to solve their doubts. After the project was completed, instructors graded the submitted solutions based on code correctness and maintainability. Incorrect code was detected by the automatic testing system, so instructors focused solely on code maintainability. Incorrect code was graded with a score below 5 out of 10.

3.3. System architecture

According to Dillenbourg and Tchounikine (2007), two different levels can be identified in a learning script. First, at the *macro-script* level, the activity flow consists in a predefined sequence of activities that produces the desired interactions with the course material. Second, the *micro-script* refers to the scaffolding of the interaction *per se*. In the presented case of study, the *macro-script* refers to the three sequential assignments in which the project is divided. The *micro-script* consists in the support for the completion of the assignments: the students have to submit a solution to the problem (a file with source code) and they instantly received the results of the correctness tests and the corresponding feedback.

While the existing frameworks for the use of learning scripts focus on supporting *macro-script* (e.g. IMS LD claims to allow the use of a wide range of pedagogical models), they do not provide the functionality nor the flexibility expected for the scaffolding of *micro-scripts*, where case specific tools would better support the task. In our case of study, since IMS LD provides no tool to check code correctness, an external tool is used. GSI allows the bidirectional exchange of information between the IMS LD server and the external tool, making possible the adaptation of the activity sequence depending on the external tool's data. In other words, GSI allows IMS LD to mark the activity as finished when the external tool says that the submitted solution is correct. It follows an overview of the used architecture, a complete description of the GSI model can be found in (de-la-Fuente-Valentín et al., 2011).

The architecture of the orchestration system is summarized in Fig. 2. At the macro-script level, that is, within the IMS LD run time environment, the course participants access the player to obtain the instructions for their next activity. Their input (*access the content*, *submit a solution*, etc.) produce the *events* that trigger the IMS LD actions. Such actions can be locally processed (e.g. set the activity state as *submitted*) or they may require data from the external tool. In the latter case, GSI retrieves the required information, which is used to complete the corresponding action.

At the *micro-script* level, there are two possible interaction types: first, students access the third party application via the user interface provided by the tool. Second, the IMS LD player exchanges information with the tool in the background. From the point of view of the IMS LD player, the case specific tool is a black box that informs when a certain action has been executed. From the point of view of the students, the case specific tool provides the functionality required to properly execute the proposed activity.

3.4. Formalization of the scripted flow

In order to be enacted within the described orchestration system, the scripted flow detailed in Section 3.2 needs to be formalized according to IMS LD, being the interactions with the case specific tool modeled with GSI. This subsection uses the IMS LD vocabulary to present the most relevant details of the formalized scripted flow. The complete script can be downloaded at http://bit.ly/progsis-UoL.

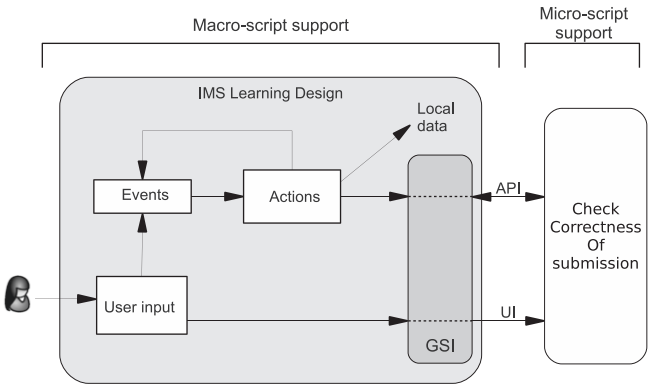

Fig. 2. Architecture of the proposed orchestration system.

From the point of view of the students, the course consists in two *activity structures*, one of *selection* type (all the activities available from the beginning) with the overview of the project; and the other one of *sequence* type (the activities are sequentially unlocked) with the specific assignments. Two conditions are imposed on the assignments to be unlocked: first, the students must submit a correct solution; second, the students must not have been penalized. This fact is modeled in the script with *properties* and *conditions*, so that the value of an *activity-N-Finished* property says whether the submission is correct or not. Such correctness is calculated and stored in the external tool, so the IMS LD engine has to request such information and store it in a *local-role property*. Such interaction is formalized in the script by combining IMS LD and GSI namespaces. Listing 1 shows the XML code that retrieves such information.

```
<!–
- The XML is equivalent to:
- if (!activity One Finished)
- activity One Finished = get ExternalValue(serviceref="…", owner="self", position = "1")
- –>
<imsld:if>
<imsld:is>
<imsld:property-ref ref="activity One Finished"/>
<imsld:property-value>0</imsld:property-value>
</imsld:is>
</imsld:if>
<imsld:then>
<imsld:change-property-value>
<imsld:property-ref ref="activityOneFinished" />
<imsld:property-value>
<gsi:external-value>
<gsi:contribution-value serviceref="delivery-platform-service" owner="self" position="1"/> </gsi:external-value>
</imsld:property-value>
</imsld:change-property-value>
</imsld:then>
```

Listing 1
"Conditions used in the IMS LD manifest to retrieve external values"

From the teacher perspective, the course is an *activity structure* whose activities are presented in *selection* mode. With such structure, the teachers have access to all the course material at once, and they can also edit some properties of the course. More specifically, teachers can modify the deadlines set for the student submissions. Since each date is modeled as a *local property*, the resource to modify their value has *imsldcontent* type.

4. Evaluation methodology

The objective of the evaluation is to analyze students performance and instructor workload with the use of the described orchestration system, and compare these parameters with the previous course edition in which no such scaffolding was provided. More specifically, the evaluation was guided by the following propositions:

P1) The students will maintain a sustained effort.
P2) Instructor workload will be similar to previous course editions.
P3) A reduction in the drop out rate will be detected as a consequence of students being more engaged in the course than in previous editions.
P4) An increase in overall academic performance in the course will be detected also as a consequence of students being more engaged in the course.

These propositions were analyzed following a mixed method evaluation technique (Martínez, Dimitriadis, Rubia, Gómez, & de-la-Fuente, 2003). Mixed methods combine quantitative techniques, such as closed questions, with qualitative techniques such as open questions, discussion groups or observations. There is an inherent subjectivity on the qualitative analysis and, in order to increase the validity of the findings, a *triangulation* method was used to reinforce each of the interpretations extracted through a comparative analysis of evidence provided from different sources.

The data under analysis was obtained from three sources summarized in Table 1. First, the final course grades. The software project is expected to influence over the rest of the graded tasks. In order to measure such impact, it would have been desirable to create a control group in which the students do not use the proposed system. However, this approach was not possible due to administrative and ethical issues. Instead, the analysis considered all the graded activities in the 2008/2009 and 2009/2010 editions of the course.

The second data source is a survey that students were asked to fill. The survey was anonymous, voluntary, and had no effect in the grade. Students could answer until the final examination. The survey contained open-ended questions, multiple choice and five-point Likert scale question. The objective was to capture the student opinions of the scaffolding method. A report of the survey results can be downloaded from http://bit.ly/blind-author-progsis-report.

Additionally, the teaching staff filled a survey after the final examination was completed to gather their qualitative opinion of the scaffolding method. The questions were both Likert scale and open ended.

5. Results

5.1. Student workload and sustained effort

The first proposition stated in Section 4 is aimed at evaluating to what extent the scaffolding method promoted sustained effort during the course. That is, if the students are devoting time to the project and if they do so in a day-to-day basis.

Fig. 3 shows student activity in the platform performing the orchestration. The six submission deadlines can be clearly identified, but it can also be seen that the system received submissions almost every day, except during the time between the last deadline of the first project and the publication of the second project. Furthermore, the peaks in the "access to the course content" line shows that the platform delivered the course material at least 100 times a day. Table 2 shows that 90.35% of the received submissions passed the test cases, where the 92.8% of such correct submissions were received before the deadline. The table also shows that some of the teams were not able to develop a correct solution in time, but they kept working and submitted it after the deadline.

Table 3 shows the time that students dedicated to the project as reported in the survey. A total of 30.77% and 54.81% of the students worked more than 20 h on the first and second project respectively. Taking into account that there were 6 tasks to be completed in a 12 week period, it can be concluded that students maintained a sustained effort in almost every week of the course.

The time devoted to complete the projects doubles the expectations stated in Section 3. Accordingly, students identified these projects as the most time-demanding task during the course and suggested to increase their relevance in the final course grade. A large number of students stated that *"we have really worked a lot of hours in the project"* and that *"the projects required too much time for its weight in the course grade"*. Despite the required effort, the survey shows that students valued the relevance of the continuous work within the course. For example, one student said that *"there is no problem if you keep your work up to date"*, and another student stated that *"this method forces you to finish the task on time, which helps people to keep tasks up to date"*.

The workload imposed on the students was also reflected in the intensive use they did of the office hours with the instructors. Students asked questions about the project more frequently than in previous years, both in laboratory sessions and during office hours. Instructors were asked about their perception of sustained effort of the students and, in a scale from 1 (unsustained effort) to 5 (very sustained effort), their answers averaged 3.88. In fact, some instructors perceived an excessive workload imposed on the students. For example, one instructor said that *"two projects represent too much work, we should consider having only one project in the future"*. The interesting fact about this opinion arises when the course is compared with previous editions, where the difficulty and size of the projects were similar but the instructors did not perceive such excessive workload. It can be argued that the scaffolding method really forced the students to work, and instructors perceived more clearly the actual workload derived from the projects.

Despite the excessive workload, students valued the scaffolding method as something beneficial. Fig. 4 shows that 2 out of 3 students would like to use the system in future courses, while only a 15% of students disliked the system. Improvements to the system were suggested in the survey, the most demanded one can be summarized with this opinion: *"I would add the possibility of repeating the submission once the code has passed the test, so that we can improve the code and its comments"*.

5.2. Instructor workload

The tasks required from the teaching staff were the following: first, they had to explain to the students the methodology of the project and the requirements of the application. Then, the instructors tutored the students both in laboratory sessions and during office hours. Instructors were also responsible for setting the submission dates, so they had to be coordinated with instructors from other groups. Finally, they had to grade all submissions. Apart from setting the submission dates, these are the same tasks carried out in the previous course edition. The measurement of the impact of the scaffolding method in instructor workload is based on the comments and numerical answers obtained from the survey.

Table 1
Data sources for the evaluation of the experience.

Source	Description	N
Student grades	Numeric data including the final grade and the partial results	425
Student survey	Likert scale and open-ended questions, anonymous, not mandatory, and answers were collected before the final exam	104
Instructor survey	Likert scale and open-ended questions. Answers were collected after the final exam	8

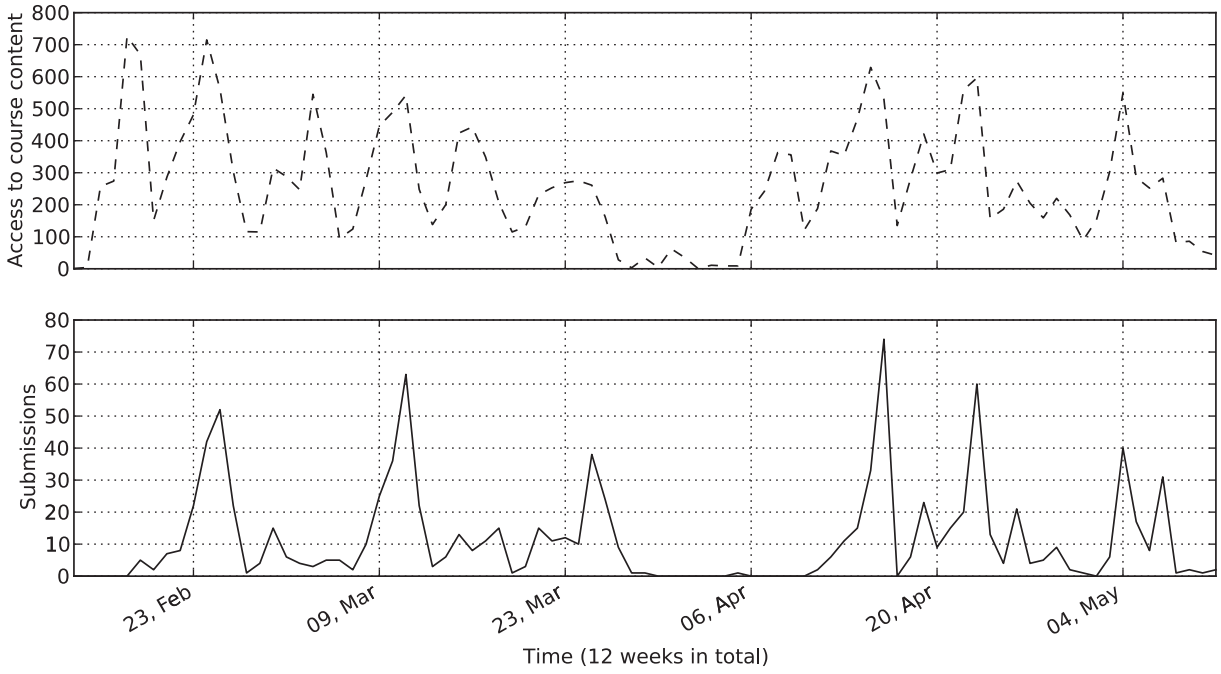

Fig. 3. Students activity in the platform.

Instructors perceived an adequate workload (3.57, where 1 and 5 meant low and high workload, respectively) and some of them stated that they have perceived less workload than in the previous edition. Regarding the scaffolding method, the following factors were identified as an improvement:

- All instructors stated that the automatic test of the submissions saved a lot of time.
- Two instructors stated that the automatic orchestration of the content saved some time.
- Other factors mentioned by instructors regarded specific features of the platform, such as the possibility to download all the submissions of a certain group in a single file.

There were also factors identified as something that increased their workload mainly related to usability problems of the platform, such as an excessive number of clicks required to access the content, or the time required to get used to the platform.

Additionally, instructors reported a more intensive usage of office hours by the students. Some instructors devoted more time to answer these questions than the time they initially scheduled for the task. There was an agreement that answering students during office hours was the most time demanding task this year: *"the semiautomatic grading system has increased the workload derived from project related questions"*.

5.3. Drop out rate

Without the existence of a control group to perform a solid comparison, it is difficult to evaluate whether the scaffolding method has contributed to a change in the drop out rate, which is the research question addressed in proposition P2). However, the presented analysis is based on a comparison between the course stats from the 2008/2009 and 2009/2010 course editions. Despite the fact that the only change in the course between these two editions was the use of the proposed scaffolding method, there might be other factors that influence the overall course results. Some student comments extracted from the survey support that the scaffolding method had an impact on the attrition rate, but it is not possible (and probably, not realistic) to assert that it was the only factor.

Table 4 contains the average scores in both editions. The *t*-test was used to evaluate the statistical significance of the means' difference in the two course editions. The table shows a significant decrease in the drop out rate: 36.23% of the registered students left the course before the final exam, in contrast to the 51.71% observed in the previous year. Regarding this fact, one positive feature of the scaffolding method was the quick feedback offered to the students: they instantly knew if they succeeded in their development, with a consequent increase of their self-efficacy. The prevailing opinion in the survey was that *"it is an incentive, because it is satisfactory to know quickly if your code works"*. Furthermore, some of the students perceived a *gamified* learning activity. They stated that *"having to clear levels is like an incentive for our motivation"*.

Table 2
Submissions received.

		Project 1	Project 2	Total
Before deadline	Pass	483	432	825
	Do not pass	35	38	73
After deadline	Pass	35	47	64
	Do not pass	8	14	22
Total		543	441	984

Table 3
Time used by students in project development.

		Project 1	Project 2
Time spent	<10 h	13.46%	8.65%
	10–15 h	37.50%	15.38%
	15–20 h	18.27%	21.15%
	20–25 h	17.31%	19.23%
	>25 h	13.46%	35.58%

Supported by the observed reduction of the drop out rate and student comments similar to the above quoted ones, it is our belief that the proposed scaffolding method engaged the students in the programming task and, despite the reported student workload, they were less likely to drop the course than in the previous year.

5.4. Overall course results

In this section, the overall course results of the two course editions are compared to see if the proposed orchestration had any effect as stated in proposition P3 in Section 4. The comparative is based on the data presented in Table 4.

The first observed result is a considerable improvement in the project scores in course 2009/2010 with respect to the previous year. The comments extracted from the survey indicates that the scaffolding method played an important role in such improvement. For example, one of the students said, *"once submitted, we know whether ours code succeeds or not"*. The immediate result is that the students were able to iterate until they got the right solution. In contrast, the students of the 2008/2009 edition only knew if their submitted solution was correct when the final grades were published, so there was no chance to improve their solution.

The second finding is a statistically significant decrease in the results in the midterm examinations (p-value < 0.05). Two factors were identified as the possible cause of such difference. First, the lack of study time because of the workload imposed by the practical tasks. A number of students complained that the project prevented them to dedicate enough time to study the theoretical aspects of the course (e.g. *"sometimes it is impossible to study, because of the lack of time"*). Furthermore, in some of the groups the project deadline and the examination dates overlapped so students decided to devote more time to the project. This behavior is coherent with the comments requiring a higher weight of the projects in the course grade.

The second plausible factor can be summarized by the following quote extracted from the instructor survey: *"the student goal is to pass the course, rather than learning how to program"*. That is, the students reached a grade to pass the course working mostly in the project and perceived the midterm exam as less relevant. This factor is also supported by the data in Table 4, where there is a statistically significant difference between the two course editions on all the partial scores, but there is no such difference in the final grade.

Finally, the course overall score was slightly higher than in the previous year, but with no statistically significant difference (p-value > 0.05). That is, those who took the course had an overall performance similar to the obtained in the previous year.

In summary, the two most significant differences between the two course editions are:

- The average total score was similar in the two course editions, but there was a difference on how the students obtained their score.
- The reduction in the drop out rate resulted in more students passing the course.

5.5. Set up workload

The course life cycle imposed by learning scripts consists of three different phases: authoring, deployment and enactment. In the first phase, the course author defines content, roles, and the rules that will drive the course flow. Then, the deployment phase is devoted to allocate all the required resources and tools. Finally, course participants interact with the content during the enactment phase. As summarized by the propositions given in Section 4, this paper aims at analyzing enactment aspects of a learning script that supports

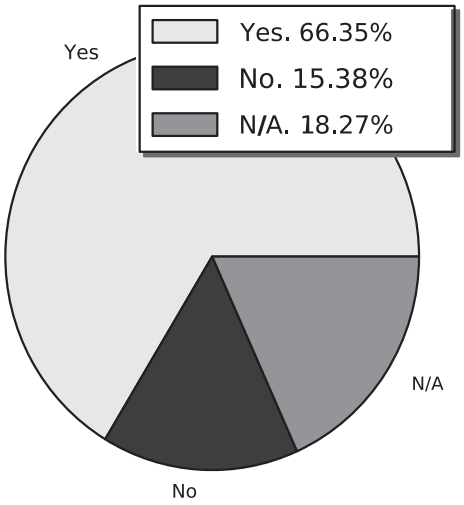

Fig. 4. Answers to the question: would you like to use the same submissions system in future courses?

Table 4
Student results in different course editions.

		2008/2009	2009/2010	p-value
Drop out rate		212/410 51.71%	154/425 36.23%	$3.0 \cdot 10^{-6}$
Project 1 score	Mean:	4.76	6.63	$2.8 \cdot 10^{-11}$
	Std:	4.00	3.38	
Project 2 score	Mean:	4.03	6.65	$2.2 \cdot 10^{-16}$
	Std:	4.00	3.06	
Midterm 1 score	Mean:	4.80	4.28	0.0036
	Std:	2.15	2.31	
Midterm 2 score	Mean:	5.00	4.14	$3.8 \cdot 10^{-5}$
	Std:	2.53	2.26	
Final exam score	Mean:	4.06	4.54	0.0152
	Std:	2.54	2.24	
Total score	Mean:	5.18	5.413	0.0965
	Std:	2.03	1.81	
Passed the course (over registered)		123/410 30.00%	180/425 42.35%	$9.7 \cdot 10^{-5}$
Passed the course (over non dropouts)		123/198 62.12%	180/271 66.42%	0.1697

students sustained effort, being authoring and deployment out of the focus of the study. However, it is worth to mention the main difficulties found during the set up of the learning script, and the imposed workload.

The authoring phase is an unsolved issue in IMS LD. The straightforward consequence is that the creation of scripts is time consuming and error prone. Additionally, the orchestration system used in the presented work uses GSI to interact with external tools. As the GSI model follows the same authoring principles, its inclusion in the system does neither increase nor decrease such design difficulties. Despite there exist some efforts devoted to ease the authoring process (Griffiths, Beauvoir, Liber, & Barret-Baxendale, 2009; Hernández-Leo et al., 2005; Neumann & Oberhuemer, 2008), none of them supports GSI. The author's decision was to create the script with an XML editor, which requires a certain level of expertise with the IMS LD specification. Having that the second project was, except for the content, equal to the first one, the simple substitution of content files was enough to create the new script. This fact suggest the use of a template based strategy (Hernández-Leo et al., 2005) in the authoring of new scripts.

One of the expected strengths of learning scripts is their reusability. That is, a learning script should be able to be deployed several times with no extra effort. In the presented case of study, the same script was separately deployed for each of the different groups. During such deployment, students had to create the teams among themselves and report their decision by certain date. There were several teams reported past the deadline, and numerous team adjustments needed to be made during the project execution. IMS Learning Design did not offer the desired flexibility to accommodate these changes which translated in a peak in managerial tasks.

In sum, the set up workload was mainly caused by the authoring of the first script, while the time devoted to the deployment stayed into reasonable margins. Such situation encourages future reuse of both the orchestration system and the learning script, due to the observed benefits in terms of enactment workload, students support and drop out rate.

6. Conclusion

This work presented a case of study where project-based learning activities in a first-year computer programming course were automatically deployed depending on students individual performance. The technology-based orchestration system had two goals: first, to reduce the drop out rate by engaging the students in the course; second, to keep instructor workload into reasonable margins in the deployment of the method in a large enrollment course. The proposed architecture is based on the combination of IMS LD and GSI: IMS LD was used to orchestrate the activity flow and GSI provided communication with a tool implementing automatic software testing.

The experimental results show that, with the proposed approach, students maintained a sustained effort during the course. They developed code every week during the semester. Additionally, there has been a significant decrease in the number of students that dropped the course and, consequently, the number of students passing the course increased. The automation of the tasks used for the project-based learning activities was well-considered by students, and the majority of them said they would like to use the same method in future courses.

Although the method had almost no impact in instructor workload, they reported an increase in student interactivity in the laboratory sessions, as well as a more frequent use of the office hours than in previous edition. With this increase in participation, instructors have obtained more precise feedback from the students, so that they realized the actual workload imposed by the project. Future versions of the course should take this feedback into account because, in spite of the observed decrease in the level of failure, such rate is still above the expectations.

The experience allowed instructors to observe the students reaction to the overall grading plan: the average total score was similar in the two course editions, but there was a difference on how the students obtained their score. Such difference suggests that the students devoted more time to practical activities than in previous years, at the expense of their study time. In the presented experiment, no indicators measured to what extent this different behavior resulted in the acquisition of different skills (e.g. a higher order understanding). Further research may clarify this fact.

The specific combination of IMS LD and GSI allowed instructors to deploy a complex scaffolding strategy and scale it to a large number of students with barely any increase in administrative duties. Thus, the proposed solution shows a real-life scenario where technology may support pedagogy to reduce the adoption barrier and fully embrace competence-based learning methods and guarantee their effectiveness beyond small groups.

Acknowledgments

Work partially funded by the EEE project, "Plan Nacional de I+D+I TIN2011-28308-C03-01" and the "Emadrid: Investigación y desarrollo de tecnologías para el e-learning en la Comunidad de Madrid" project (S2009/TIC-1650). The authors would like to thank the course

teaching staff: M. Carmen Fernández Panadero, Julio Villena Román, J. Jesús García Rueda, César Martín del Álamo, Javier Fernández Galván, José María Rubio Manso, Ben Marx, Raquel Crespo García and Jesús Arias Flisteus.

References

ACM/IEEE. (2005). *Computing curricula 2005* (The overview report).

Albanese, M. A., & Mitchell, S. (Jan. 1993). Problem-based learning: a review of literature on its outcomes and implementation issues. *Academic Medicine: Journal of the Association of American Medical Colleges, 68*(1), 52–81.

Arnow, D., & Barshay, O. (Sep. 1999). On-line programming examinations using web to teach. *ACM SIGCSE Bulletin, 31*(3), 21–24.

Berger, A., Moretti, R., Chastonay, P., Dillenbourg, P., Bchir, A., Baddoura, R., et al. (2001). Teaching community health by exploiting international socio-cultural and economical differences. In P. Dillenbourg, A. Eurelings, & K. Hakkarainen (Eds.), *Proceedings of the first European Conference on Computer Supported Collaborative Learning* (pp. 97–105).

Berkson, L. (Oct. 1993). Problem-based learning: have the expectations been met? *Academic Medicine Journal of the Association of American Medical Colleges, 68*(10 Suppl.), S79–S88.

Brusilovsky, P., Kouchnirenko, A., Miller, P., & Tomek, I. (1994). Teaching programming to novices: a review of approaches and tools. In *Proceedings of ED-MEDIA 94 – World Conference on Educational Multimedia and Hypermedia* (pp. 103–110).

Burgess, K. L. (Jan. 2004). Is your case a problem? *Journal of STEM Education Innovations and Research, 5*(1), 42–44.

Burgos, D., Tattersall, C., & Koper, R. (2007). How to represent adaptation in e-learning with IMS – learning design. *Interactive Learning Environments, 15*(2), 161–170.

Cheang, B. (Sep. 2003). On automated grading of programming assignments in an academic institution. *Computers & Education, 41*(2), 121–131.

Dillenbourg, P., & Jermann, P. (2007). Designing integrative scripts. *Scripting Computer-supported Collaborative Learning*, 275–301.

Dillenbourg, P., & Tchounikine, P. (Feb. 2007). Flexibility in macro-scripts for computer-supported collaborative learning. *Journal of Computer Assisted Learning, 23*(1), 1–13.

Edwards, S. H. (Oct. 2003). Rethinking computer science education from a test-first perspective. In *Companion of the 18th annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications – OOPSLA '03* (pp. 148). New York, New York, USA: ACM Press.

Feldman, J., & Jones, J. (May 1997). Semiautomatic testing of student software under Unix(R). *IEEE Transactions on Education, 40*(2), 158–161.

Fincher, S. (1999). What are we doing when we teach programming?. In. *Frontiers in Education Conference 1999 FIE99 29th Annual, Vol. 1* IEEE, pp. 12a41–5.

de-la-Fuente-Valentín, L., Pardo, A., & Kloos, C. D. (2011). Generic service integration in adaptive learning experiences using IMS learning design. *Computers & Education, 57*(1), 1160–1170.

Garcia-Robles, R., Diaz-del Rio, F., & Vicente-Diaz, S.,A. (2009). An eLearning standard approach for supporting PBL in computer engineering. *IEEE Transactions on Education, 52*(3), 328–339.

Griffiths, D., Beauvoir, P., Liber, O., & Barrett-Baxendale, M. (Aug. 2009). From reload to recourse: learning from IMS learning design implementations. *Distance Education, 30*(2), 201–222.

Hernández-Leo, D., Asensio-Pérez, J., & Dimitriadis, Y. (2005). Computational representation of collaborative learning flow patterns using IMS learning design. *Educational Technology & Society, 8*(4), 75–89.

Hernández-Leo, D., Bote-Lorenzo, M. L., Asensio-Perez, J. I., Gomez-Sanchez, E., Villasclaras-Fernandez, E., Jorrín-Abellán, I. M., et al. (2007). Free- and open-source software for a course on network management: authoring and enactment of scripts based on collaborative learning strategies. *IEEE Transactions on Education, 50*(4), 292–301.

Higgins, C. A., Gray, G., Symeonidis, P., & Tsintsifas, A. (Sep. 2005). Automated assessment and experiences of teaching programming. *Journal on Educational Resources in Computing, 5*(3), Article 5.

IMS Global Learning Consortium, Feb. 2003. *IMS learning design specification*. Tech. rep.

Jackson, D., & Usher, M. (Mar. 1997). Grading student programs using ASSYST. *ACM SIGCSE Bulletin, 29*(1), 335–339.

Jermann, P., & Dillenbourg, P. (2003). *Elaborating new arguments through a CSCL scenario*.

Jones, E. L. (Nov. 2000). Grading student programs – a software testing approach. *Journal of Computing Sciences in Colleges, 16*(2), 185–192.

Kinnunen, P., & Malmi, L. (2006). Why students drop out CS1 course?. In *Proceedings of the Second International Workshop on Computing Education Research* (pp. 108) ACM.

Köse, U. (2010). A web based system for project-based learning activities in "web design and programming" course. *Procedia – Social and Behavioral Sciences, 2*(2), 1174–1184.

Kurnia, A., Lim, A., & Cheang, B. (May 2001). Online judge. *Computers & Education, 36*(4), 299–315.

Martinez-Mones, A., Gomez-Sanchez, E., Dimitriadis, Y. A., Jorrin-Abellan, I. M., Rubia-Avi, B., & Vega-Gorgojo, G. (2005). Multiple case studies to enhance project-based learning in a computer architecture course. *IEEE Transactions on Education, 48*(3), 482–489.

Martinez-Ortiz, I., Sierra, J.-L., & Fernandez-Manjon, B. (Jul. 2009). Authoring and reengineering of IMS learning design units of learning. *IEEE Transactions on Learning Technologies, 2*(3), 189–202.

Martínez, A., Dimitriadis, Y., Rubia, B., Gómez, E., & de-la-Fuente, P. (2003). Combining qualitative evaluation and social network analysis for the study of classroom social interactions. *Computers & Education, 41*(4), 353–368.

McKinney, D., & Denton, L. F. (2004). Houston, we have a problem: there's a leak in the CS1 affective oxygen tank. In *SIGCSE '04: Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education* (pp. 236–239). New York, NY, USA: ACM.

Milne, I., & Rowe, G. (Mar. 2002). Difficulties in learning and teaching programming – views of students and tutors. *Education and Information Technologies, 7*(1), 55–66.

Neumann, S., & Oberhuemer, P. (2008). Supporting instructors in creating standard conformant learning designs: the graphical learning modeler. In. *World Conference on Educational Multimedia, Hypermedia and Telecommunications 2008, Vol. 2008* (pp. 3510–3519).

Nuutila, E., Törmä, S., Kinnunen, P., & Malmi, L. (2008). Learning programming with the PBL Method–Experiences on PBL cases and tutoring. In *Reflections on the teaching of programming* (pp. 47–67). Springer.

Reek, K. (1996). A software infrastructure to support introductory computer science courses. *ACM SIGCSE Bulletin, 28*(1), 125–129.

Robins, A., Rountree, J., & Rountree, N. (Jun. 2003). Learning and teaching programming: a review and discussion. *Computer Science Education, 13*(2), 137–172.

dos Santos, S., da Conceição Moraes Batista, M., Cavalcanti, A., Albuquerque, J., & Meira, S. (2009). Applying PBL in software engineering education. In *Proceedings of the 2009 22nd Conference on Software Engineering Education and Training* (pp. 182–189). IEEE Computer Society.

Spacco, J., Hovemeyer, D., Pugh, W., Emad, F., Hollingsworth, J. K., & Padua-Perez, N. (Jun. 2006). Experiences with marmoset. In. *Proceedings of the 11th Annual SIGCSE conference on Innovation and technology in computer science education – ITICSE '06, Vol. 38* (pp. 13). New York, New York, USA: ACM Press.

Stahl, G. (2005). Group cognition in computer-assisted collaborative learning. *Journal of Computer Assisted Learning, 21*(2), 79–90.

Towle, B., & Halm, M. (2005). Designing adaptive learning environments with learning design. In *Learning design, a handbook on modelling and delivering networked education and training* (pp. 215–226).

Tseng, K., Chiang, F., & Hsu, W. (May 2008). Interactive processes and learning attitudes in a web-based problem-based learning (PBL) platform. *Computers in Human Behavior, 24*(3), 940–955.

Valdivia, R., Nussbaum, M., & Ochoa, S. (Aug. 2009). Modeling a collaborative answer Negotiation activity using IMS-based learning design. *IEEE Transactions on Education, 52*(3), 375–384.

White, G., & Sivitanides, M. (2002). A theory of the relationships between cognitive requirements of computer programming languages and programmers' cognitive characteristics. *Journal of Information Systems Education, 13*(1), 59–66.

Wiedenbeck, S. (2005). Factors affecting the success of non-majors in learning to program. International Computing Education Research Workshop, 13.

Yadav, A., Subedi, D., Lundeberg, M. A., & Bunting, C. F. (2011). Problem-based learning: influence on students' learning in an electrical engineering course. *Journal of Engineering Education, 100*, 253–280.

Zhang, J., & Du, H. (Dec. 2008). The PBL's application research on prolog language's instuction. In *2008 International Workshop on Education Technology and Training & 2008 International Workshop on Geoscience and Remote Sensing* (pp. 112–114). IEEE.