IMPLEMENTING AN INCREMENTAL PROJECT-BASED LEARNING SOLUTION FOR CS1/CS2 COURSES

Carlos Vega, Camilo Jiménez and Jorge Villalobos
Systems and Computing Engineering Department, Universidad de los Andes, Bogotá, Colombia

Keywords: Student-centred Learning, Active Learning, Incremental Learning, Project-based Learning, Programming Skills, Education.

Abstract: Cupi2 is a project that promotes an integral solution to problems in teaching/learning programming using a large and structured courseware, and a student-centred pedagogical model (Villalobos and Casallas, 2006a; Villalobos, Calderón, and Jiménez, 2009a; Villalobos, Calderón, and Jiménez, 2009b; Villalobos and Jiménez, 2010). As a cornerstone of Cupi2, we use incremental projects intended to motivate students, and to develop high-level programming skills throughout their learning. A critical factor of these projects is that they are specially designed so that students are engaged in activities that complete a scaffold of a complete program. However, both the scaffolds and the activities needed to complete these incomplete programs must be arranged carefully by the instructors in order to stress the adequate contents for students, and at the same time, to help those students acquire programming skills effectively. Jointly, scaffold versions need to comply with high quality standards, representing a high time consuming activity for instructors, and therefore, increased costs for institutions. In this paper, we describe the way we overcome these challenges by supporting the projects' design in a scalable way with a software factory.

1 INTRODUCTION

Learning and teaching computer programming courses has been a challenge for higher education institutions for the past twenty years (Baeten et al., 2010); (Lea et al., 2003). Students perceive programming as a hard subject, and it is common to hear recurrent issues related to students' frustration and lack of motivation. Opinions like "I do not feel like I belonged", "classes were unfriendly", or "classes were boring" are very common among students (Biggers et al., 2008). On the other hand, programming courses have been reported to have several methodological problems related with the approach instructors adopt to teach (Cannon and Newble, 2000). Particularly, they have ignored that learning to program well, is much like learning to write: Students need to understand the intention, receive detailed feedback, rewrite and receive more feedback. Instead, students are confronted with lectures that explain non-contextualized contents (e.g., programming fundamentals and algorithms), rather than being confronted with activities to generate programming skills applicable in different situations (Woodley and Kamin, 2007). As a consequence, students have acquired a sense that "how to do/apply things" is not teachable, and is rather something that depends on inspiration and on the genius of the programmer. They usually do not see the real need for concepts at all, while becoming anxious about it and making their learning process harder in the process (Villalobos and Casallas, 2006a).

In order to overcome these challenges during computer programming courses, we designed an integral learning approach called Cupi2 (http://cupi2.uniandes.edu.co) (Villalobos and Casallas, 2006a); (Villalobos et al, 2009a); (Villalobos et al, 2009b); (Villalobos and Jiménez, 2010). This approach is based on a concrete pedagogical model we call incremental and project-based, since students have to work completing and extending projects incrementally through different levels of mastery. Projects are contextualized real-world incomplete programs that students have to scaffold out of the classroom throughout each level. This way, they are motivated when they apply their knowledge in different real situations, while developing programming skills incrementally. Overall, students work on 12 projects among two

DOI: 10.5220/0003900500150027
In *Proceedings of the 4th International Conference on Computer Supported Education* (CSEDU-2012), pages 15-27
ISBN: 978-989-8565-07-5

highly attended basic programming courses: CS1 and CS2. Approximately 1.250 students attend these courses, and more than 40 instructors are needed each semester. Jointly, several universities in Colombia have adopted this model, augmenting the number of students that use Cupi2 to more than 2.000 each semester.

In that scale, supporting such a project-based pedagogical model becomes a challenging task. In particular, building projects that stress similar contents and motivation to each student, and under high quality standards becomes a time consuming activity for instructors and highly costly to our institution. Not to mention that an excellent design and construction is also essential. This is not only the case from a software development standpoint, since students are scaffolding incomplete programs, but also from a pedagogical perspective, since the projects are one of the cornerstones of our approach. In this paper, we present how we support our pedagogical model with a software factory for building such projects. First we describe the pedagogical impact of the projects in our learning approach, and then we show how a set of methodologies and structured tools make our pedagogical model sustainable. We also discuss some findings around our experience with this project-based learning approach, and we describe its applicability in other contexts apart from our University as well.

Throughout the last six years, this incremental project-based learning approach has shown successful results at our university. Firstly, the number of students who fail computer programming courses has declined up to 50%. Secondly, the results of evaluations made by students about their perception of our computer programming courses have increased positively by more than 30%. Furthermore, the students' average grade in computer programming courses has increased by more than 16%. These results show a general improvement in computer programming courses, and they also reflect the increase of learners' satisfaction for programming and the decrease of drop-out rates in our programming courses.

Cupi2 has also been recognized by important regional institutions in two different occasions. In the first occasion, Cupi2 was awarded the 2007 Colombian Informatics Award by the Association of Colombian Computer Engineers (ACIS), based on the quality of its learning objects and its academic impact in more than 30 universities in Colombia. In the second occasion, Cupi2 obtained the first place in the 10th prize of Educational Informatics 2009 by the Iberoamerican Network of Educational Informatics (RIBIE), based on its academic and research quality, its social incidence, and the number of students and faculty members benefitting from it.

2 CUPI2: AN INCREMENTAL PROJECT-BASED LEARNING SOLUTION

Learning using projects or Project-based Learning (PbL) is about generating interest and motivation in students (Lam et al., 2009). Students are engaged in activities that are designed to either answer a question or solve a problem reflecting real life situations, while they also integrate topics from various study fields. They are continually involved in problem solving and decision making tasks and at the same time encouraged to become autonomous in their learning process.

2.1 The Projects

PbL emphasizes students' learning activities around projects (Köse, 2010). Projects are learning objects that require a question and a problem to direct activities that will result in the construction of a product or an artefact (Blumenfeld et al., 1991). This process can involve the improvement of a product as well. However, the product must always reflect a real-world like problem.

In our learning approach, projects are based in the notion of complete programs as the products students will construct. A complete program is a working computer program with an attractive graphical interface, a well-defined set of requirements, an architectural design document, a set of unit tests, the corresponding compilation and building scripts, a well-documented code, and a demonstration video. With this, we permanently encourage students to integrate and clearly discern elements from different thematic axes in programming (Villalobos and Casallas, 2006a). We realized that these axes, which are important for programming, were reviewed superficially or ignored in traditional programming courses, and for this reason, students used to end up with the wrong comprehension of programming: They used to place much more importance on the programming language rather than on the process of building a program. Contrary to that, we teach students that programming is the balance of several domains (Villalobos and Casallas, 2006a); like algorithmic, software processes, programming languages, tools, etc. (See Figure 1).

In order to construct these complete programs inside a project, students are engaged in activities to complete a partial version of them. In our words: the problem students are confronted with, is improving a skeleton of a complete program following an assignment guide. Students never lose the whole picture of a complete program. Basically, we show them that it is not necessary to cover many topics or reach the end of the course before being able to construct something interesting. This way, students have the feeling that the topics they are working on, whether they are simple or not, do have a real value in the course as they see how they apply them in the construction of complete programs throughout the course.

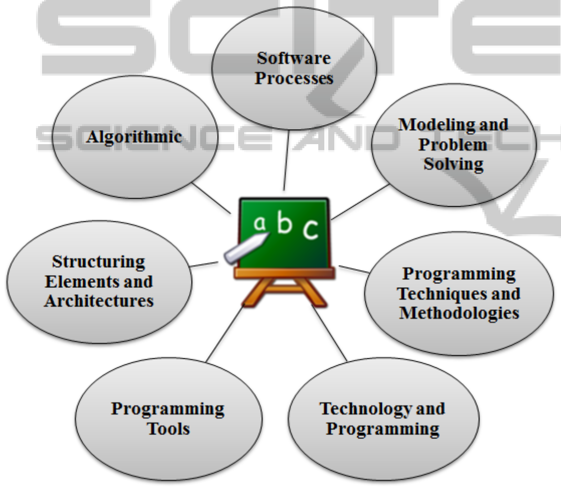
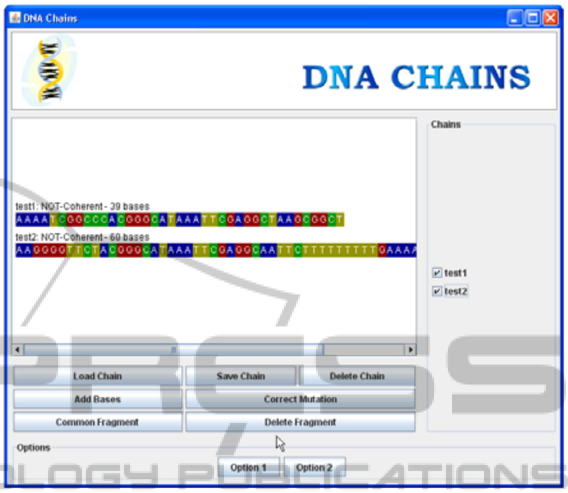

Software Processes
Algorithmic
Modeling and Problem Solving
a b c
Structuring Elements and Architectures
Programming Techniques and Methodologies
Programming Tools
Technology and Programming

Figure 1: Different domains in computer programming.

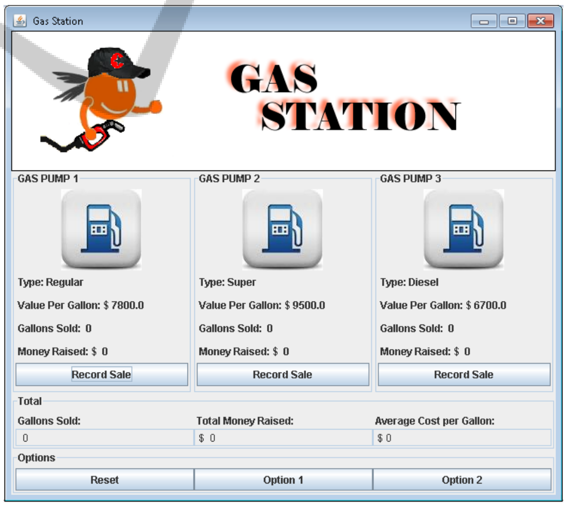
Complete programs are designed to deal with real-world-like problems. With this, they are not only meant to be motivating from a student perspective, but also from a professional one. We are aware that the needs of graduates have changed. They now need the flexibility to adapt to different professions and the ability to apply their knowledge to a wide variety of situations (Lea et al, 2003). For this reason, projects in our learning approach deal with quandaries derived from fields such as biology and mathematics, as well as with issues concerning engineering and business administration (see Figure 2 for a complete program derived from a biology problem, and Figure 3 for a complete program derived from a business administration problem).

Together with an assignment guide and a skeleton, projects include a grading matrix for instructors. We have always considered that it is really important to also recognize the learning/teaching aspects from the teachers' point of view and not just from the students' perspective. For this reason, teachers are assisted with a document that assigns a grade to the tasks students must accomplish in the project.



```
test1: NOT-Coherent - 39 bases
AAAATCGGCCCACGGGCATAAATTCGAGGCTAAGCGGCT
test2: NOT-Coherent - 60 bases
AAGGGGTTCTACGGGCATAAATTCGAGGCAATTCTTTTTTTTTGAAAA
```

Figure 2: The DNA Chains editor.



GAS PUMP 1	GAS PUMP 2	GAS PUMP 3
Type: Regular	Type: Super	Type: Diesel
Value Per Gallon: $ 7800.0	Value Per Gallon: $ 9500.0	Value Per Gallon: $ 6700.0
Gallons Sold: 0	Gallons Sold: 0	Gallons Sold: 0
Money Raised: $ 0	Money Raised: $ 0	Money Raised: $ 0

Gallons Sold:	Total Money Raised:	Average Cost per Gallon:
0	$ 0	$ 0

Figure 3: The gas station mini ERP program.

2.2 Levels

Our learning approach aims principally at generating an adequate set of programming skills in students. These skills are abilities developed to apply mature knowledge effectively in different professional domains. Figure 4 illustrates these skills organized in a mental process that students follow when solving a problem.

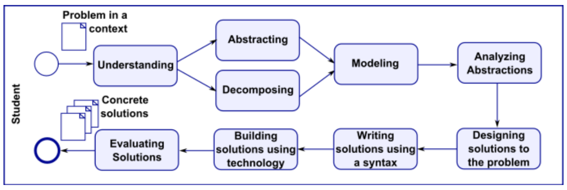


Figure 4: High level skills in computer programming.

However, teaching these high-level skills in computer programming along with the adequate programming fundamentals is a challenging task. These elements must be introduced in a gradual way so that the balance between their complexities can be preserved. To accomplish this, we defined the notion of levels. Levels help us introduce new concepts gradually, and facilitate incremental learning. This way, students do not have to reformulate their basic knowledge. Rather, they are to refine and reinforce it continually (Reinke and Michalski, 1988).

Each level is organized around a set of pedagogical objectives intended to (1) introduce or reinforce some knowledge, at the same time they are (2) generating or strengthening a set of skills. In particular this means that we permanently enforce both skill generation and programming concepts. For instance, in table 1 we list some of the pedagogical objectives for the first level.

Table 1: First level pedagogical objectives.

Id	Description
PO1-01	Understand the process of solving a problem using a computer program
PO1-02	Build a model with the elements involved in a problem and specify the services that the program should provide
PO1-03	Use simple assignment statements with arithmetic operators
PO1-04	Use simple data types such as int, double and String
PO1-05	Use methods to initialize attributes

Programming courses in Cupi2 (CS1 and CS2) are divided in 12 levels, each one during no more than three weeks. Within a level, students are engaged in a specific project. Particularly, they are given (1) an assignment guide to build a complete program, (2) a skeleton of it to be completed, and (3) a demonstration video showing its functionality. Though we design these projects carefully to encourage students in accomplishing the pedagogical objectives of each level, and to captivate them in a contextual practical experience, a proper learning environment must also be provided. We complement the project development with an active learning environment composed of different kinds of activities, such as classroom activities, extra class activities, collaborative and individual laboratory practices, and homework assignments (Villalobos et al, 2009b). Each one of these activities is configured by instructors using a vast courseware, available online through a learning community called the Cupi2 Community (Villalobos et al, 2009a). This courseware includes more than a 100 examples, 1.000 working sheets, 15 tutorials, 200 videos and animations, 35 interactive learning objects and 30 mind maps. This way, instructors are able to design activities using the most appropriate resources.

Once the course level is finished, students must submit the results of the project (a complete program) using a specific Learning Management System (LMS). Then, they are evaluated in three different ways: A revision of the project, a theoretical exam, and a practical exam. Since we are interested in assessing skills acquisition, these evaluation elements are based on two assessment mechanisms:

1) Completion: The main mandatory activity for students while working in the skeleton is to complete coding. To do so, they need to understand an existent program (i.e., its models, specifications and code). Thus, when we revise the project we are evaluating the student's skills to read code, to understand specifications and to solve problems.

2) Extension: By means of proposing extensions for students to develop, we assess their skill to abstract and apply knowledge in other contexts. We use extensions in two evaluation elements: The theoretical and the practical exam. On the one hand, theoretical exams are written evaluations about the project. Instructors must confront students with abstraction problems derived from what they worked on during the level. On the other hand, practical exams are evaluations in a computer laboratory that extend the functionality of the project.

A Cupi2 management committee fabricates the projects for each level. However, it is the responsibility of each instructor to prepare the theoretical and practical exams, and to revise the project of each student using the provided grading matrix.

Courses in Cupi2 highly depend on our definition of projects. We ground our approach in the belief that when a student develops his project, he acquires a set of skills that helps him taking the exams successfully. Therefore, it is essential that projects are well designed, implemented, and tested, both from a pedagogical and a software perspective.

3 DEVELOPING PROJECTS WITHIN A SOFTWARE FACTORY

When instructors were in charge of gathering their own supporting materials, students used to manifest several differences in their learning paths, and our Computer Science (CS) department always struggled in trying to take control of this situation. It was common to find some materials that did not develop all the course contents and others that were not interesting enough to motivate students. Similarly, the difficulty between these heterogeneous materials was not equally measured and their quality was really precarious.

In order to avoid this situation, we standardized the construction of learning materials in basic programming courses. Particularly with projects, we created a software factory that automates their development, by following a predefined process and by reusing assets such as repositories, frameworks, metrics, models, standards and tools. This factory is supported by in house tools to facilitate the management and planning of projects and its developers are qualified graduate assistants that are enrolled in the Master of Science in Systems and Computing Engineering program in our university.

By taking off the responsibility of developing projects from the instructors we discerned the possibility to scale these learning materials for any university in our local community. Besides, instructors have reported to have more time to concentrate in the instruction of programming, and students have shown better results.

3.1 Specifying the Problem

The design of a project begins selecting the target course and the level of mastery (i.e. CS2, Level 9). Since each level specifies certain pedagogical objectives, we know the kind of knowledge and skills students should be able to develop with the project. Jointly, we think of a simple but interesting real life problem, with which we can satisfy all identified educational needs derived from the pedagogical objectives. As our programming courses are taken by students of different careers, the problem usually relates to a domain different from computer science, such as finance, biology, music and others. Cases of problems in these domains are collected from instructors of other disciplines that have joined us as clients of the final program. Together, we have created a repository of cases that is being updated each semester. Such cases should be challenging, but not impossible to solve. At the same time, they must consider the level of mastery students have acquired for a specific level.

To this point, we will use a CS2 project as an example. It is called the DNA Chain Editor (see Figure 2a). It allows manipulating and visualizing simple DNA chains, and it is intended for level 9. As the pedagogical objectives state (See table 2), instead of handling arrays of DNA sequences, the DNA Chain Editor must handle linked structures in which DNA sequences are tied to their successors. On the other hand, according to PO9-3, the DNA Chain Editor must use input dialogs with several choices (Radio Buttons).

Table 2: Fourth level pedagogical objectives.

Id	Description
PO4-01	Use linked lineal structures to model groups of attributes
PO4-02	Write algorithms to manipulate linked lineal structures
PO4-03	Build GUIs using Dialogs, Radio Buttons and new Layouts

After this little analysis, the problem must be consolidated in 3 elements: A description document, a set of functional and non-functional requirements, and a conceptual model that describes the problem's entities and their relations. These elements are then validated against some metrics we have gathered in a repository of previous projects. These metrics establish the minimum and maximum number of functional requirements for a given level and they are continually updated considering also the feedback of students throughout our experience. If the mismatch with our metrics is very large, we adjust the problem to meet the defined standards for the level, so the complexity between projects of the same level is always kept similar over different semesters.

3.2 Designing the Complete Program

When the problem is consolidated and validated, we can start designing the complete program of the project. Programs in our approach must also be built in a similar way. It is important to recall that we do not want students to reformulate their basic knowledge every time they change a level. Besides, we want to permanently exemplify adequate architectures and best practices to build software. For this reason, we follow a reference-architecture (see Figure 5) that is complemented with a set of software and coding patterns.

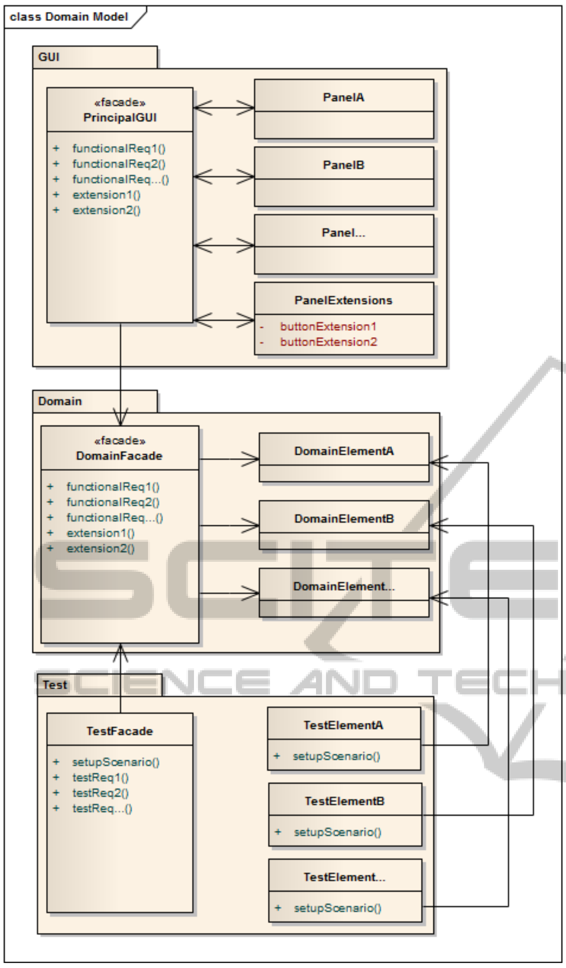


Figure 5: Reference architecture for complete programs.

The reference architecture of a complete program discerns three levels: a graphical user interface (GUI), a domain model, and a test model. Each of these levels defines a façade that communicates with elements from other levels. An MVC (Model-View-Controller) pattern is also used in the architecture of a program. Particularly, a controller that extends the JFrame element in Java-Swing must be defined. This controller implements the main method of the program and contains both a reference to the main class of the domain model (the Model in a MVC pattern) and to a set of panels that structure the GUI (the Views in a MVC pattern). This way, it controls the behavior of the application in a decoupled way separating the behaviour of the program and the user interaction behaviour.

Complete programs define a set of extension points in their architecture. Particularly, a panel with two buttons must be implemented as well as the corresponding methods to control their behaviour: One method in the controller, and another in the model. At first, this behaviour is simple: It only displays a message saying “This is an extension”. The purpose of this is that when students complete the program in their projects, they will be evaluated by extending the functionality of the program using these extension points. They are asked to change only this behaviour, for instance to traverse a certain data structure or to display data about the program.

For every class in the domain model level, there is a class that implements a set of test cases for its behaviour. It implements at least one test case for each method.

We use UML class diagrams to document the design of the complete program. Together with the corresponding class diagram, we identify the test cases for the program. They are documented in a template word file, to be used as input in the implementation phase.

Similar to the previous step, the designs are validated against previous design metrics that we have gathered in our repository. These include the number of classes in the domain, the graphical interface and the test cases, the number of attributes of each class, and the number of relations between them. Again, we adjust the design to meet the defined standards for the level in case the mismatch with our metrics is very large.

3.3 Implementing the Complete Program

To implement the complete program, we use a tool called “Application Builder”. This tool generates a base application that conforms to the projects' reference architecture from which we can proceed to build the solution (See Figure 6). It generates the following elements:

- A structured project for eclipse IDE;
- Files to run both the application and the unit tests. It generates the corresponding files for Windows (.bat) and for Mac (.sh). (In total 14 executables);
- Template files for documenting the description of the problem, specifying the functional and non-functional requirements, and sketching the UML-based domain model;
- A basic GUI: This user interface consists of three elements implemented in different classes: A basic JPanel, in which the graphical elements of the program should be included. A JPanel for extensions that consists of two buttons that will be extended in the practical exam of each level. And a JFrame that contains both panels and a reference to the principal class of the domain model;

- A basic domain model: The tool generates the principal class of the domain model intended to be used as a façade to externalize the behaviour of the application domain;
- A basic test model: A class for unit testing is also provided. This class has a basic scenario set up for the principal class of the domain model;

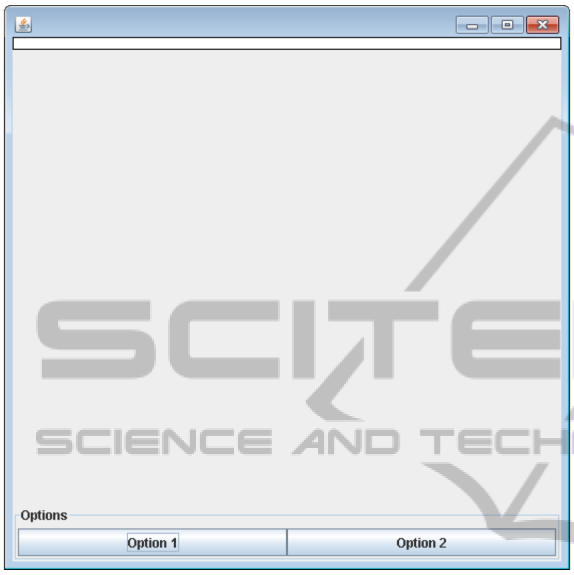


Figure 6: Generated GUI from the Application Builder.

With this base application, two groups of developers start the implementation of the program. One group is responsible for completing the GUI and the domain model classes. The other, is responsible for implementing the corresponding test cases. When both teams finish their task, they revise the work of the other group to later run the test cases. In case of errors, developers report them using an in-house tool called ChangeSet. This is a change management tool that controls the development process of software projects. Similarly, to plan the tasks associated to software changes, we use another in-house tool called Planning Tool, which is an extension of DotProject deployed as an Eclipse plug-in.

Another validation task is performed after implementing the complete program. In this case, we compare the lines of code (LOC) that result from the implementation with LOC metrics from programs built in previous semesters. If the mismatch is too large, we first try to reduce the LOC without changing the design; for example, only reducing the LOC used to implement different algorithms. If this is not possible and we need to change the design, we have to validate it against the corresponding design metrics again, before implementing the changes in the code.

3.4 Demonstration Video

To motivate students, we build a multimedia instructional video that shows how the application would work once they have successfully completed the skeleton. The video is a tour of all the features of the application, designed for students to figure out the usability and utility of it. At the same time, it encourages students to think about how to complete the skeleton.

3.5 Designing the Skeleton and the Grading Matrix

Once we have completed the functional application, we proceed with making the skeleton. Basically, we suppress parts of the application, such as source code or design documents, considering the pedagogical objectives. To do so, we rely on historical statistical data to comparatively evaluate the complexity of projects (in terms of size of requirements, application classes and lines of code to complete). We work continuously until we achieve a stable and balanced version of the skeleton. Once it is finished, we review the suppressed parts and build the assignment guide. This document must direct students to the development of the complete program and it is composed of the following parts: (1) Objectives, (2) setup, (3) development process and (4) validation process.

As for instructors, we assist them in the evaluation process of projects with a grading matrix that provides grading parameters to assess whether the student met the educational objectives proposed for the project. Using this template, we ensure that all students are evaluated under the same criteria regardless of the course section in which they are enrolled.

3.6 Inspecting the Project

When we finish the project building phase, we initiate the internal project inspection, which is made by developers of the software factory. Particularly, they must validate that the project complies with our reference architecture as well as with the documentation and coding standards. After this, we validate the project from a pedagogical perspective: We review that the educational needs (knowledge and skills extracted from the pedagogical objectives) for the chosen level are reflected in the proposed solution of the problem and the skeleton. Each of these steps is performed according to an internal review checklist.

Once the project successfully approves the internal inspection, we begin inspecting the project from an external perspective. This inspection is performed by instructors who must have two roles in mind: The teacher role and the student role. In particular, the instructor must check whether developing the project would generate the intended level skills, and at the same time, he must validate the student's ability to resolve the skeleton. He also checks that the assignment guide is clear and concise, and that it should guide the student adequately through the entire process of completing the skeleton. In this inspection, we ask instructors to do the project while reviewing an internal checklist.

4 FINDINGS AND PEDAGOGICAL REFLECTIONS

One of the big changes we have experienced with the learning approach we adopted six years ago is that we were able to increase the scope of the programming courses at the same time students obtained better results in their exams. In the last level of current CS1 for example, they end up building simple standalone versions of applications such as the Sudoku, the Excel sheet, and the Facebook (see Figure 7). In the case of CS2, students build distributed applications that communicate through sockets and use databases and SQL. It is important to mention that in the last levels of CS1 and CS2, students do not use a skeleton to build a complete program. They only receive the assignment guide that conducts them through the construction of the complete program from scratch. In previous versions of our courses (2005), students only managed to build one simple program without a graphical interface throughout the entire programming course (see Figure 8). The interaction with the user was limited by a console and it used to have approximately 60% less programming concepts compared to the projects they build nowadays. In average lines of code (LOC), this difference increases even more. Students used to build programs of one hundred LOC, 6% of the LOC they write in current projects (1.500 LOC in average).

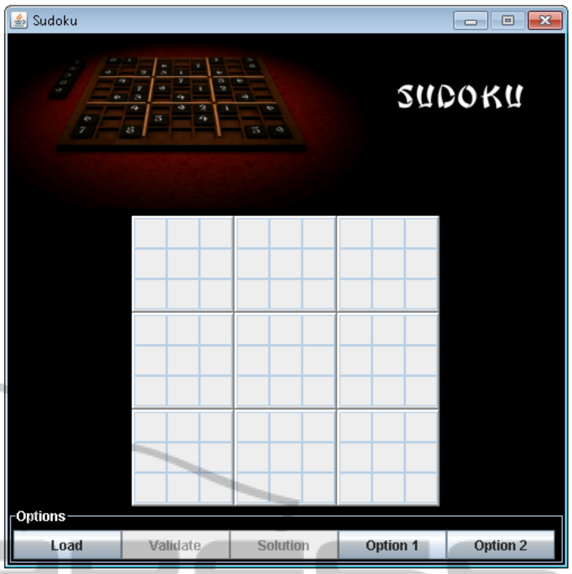
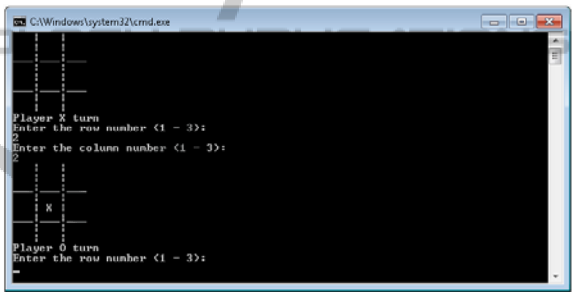


Figure 7: CS1 level 6 Project (2010): The Sudoku.



```
   |   |
___|___|___
   |   |
___|___|___
   |   |
Player X turn
Enter the row number (1 - 3):
2
Enter the column number (1 - 3):
2
   |   |
___|___|___
   | X |
___|___|___
   |   |
Player O turn
Enter the row number (1 - 3):
_
```

Figure 8: CS1 Final Project before Cupi2 approach (2005): A tic-tac-toe console version.

In other conditions, we would have suggested that this scope increase would boost the number of students who disapprove the programming courses, or even discourage them to take them. However, in our CS1 and CS2 courses the number of students who disapprove the course has fallen by 50% during these years (see Fig. 9). Based on this situation, we have noticed that it is not the complexity of the tasks or the challenging situations the only aspects that affect students' performance. On the contrary, it is their motivation and the willingness to assume these challenging situations, the aspects driving their performance, and consequently, their learning. To better support this hypothesis, our courses show an increase in the motivation of students in programming of more than 20% according to qualitative surveys performed by our University. They usually agree that "projects promote learning" and that the available courseware "facilitates the preparation for the exams".

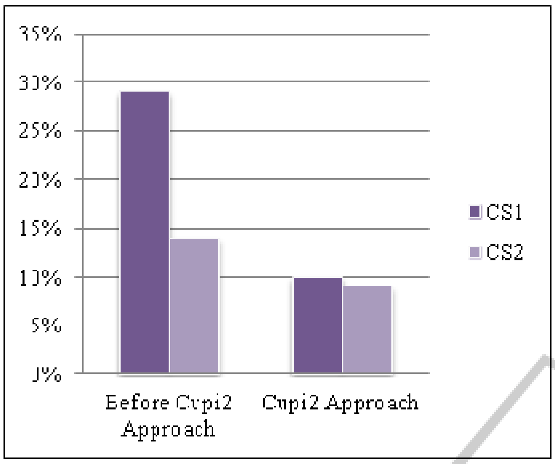


Figure 9: Students who disapprove the course in CS1 and CS2.

Not only the number of students who disapprove programming courses decreased, but their average grade in these courses increased (See Figure 10). Particularly CS1 reported an increment of 16% in the students' grade average from 3.4 to 3.92, at the same time CS2 increased this value by 7% from 3.65 to 3.89. It is important to mention that the maximum grade in these courses is 5. Similarly, the perception of programming courses also increased. Increasing by 13% and 4% respectively (See Figure 11), CS1 and CS2 courses are better appreciated than before. We now hear students saying "I used to hate programming before taking CS1", or "I did not know programming had this scope in society". In fact, it is common to find students taking programming courses as an optional course in their careers. For this reason, we have opened special programs for them at our University, so they can emphasize their current career with computer programming as something we call an "optional curriculum" within their corresponding major.

After obtaining these promising results we were not completely certain that they were caused by the adoption of our incremental PbL learning approach. Thus, we validated this hypothesis by measuring the impact the projects have in the evaluation of students. We found that 89% of those students who treated the project successfully, also succeeded in the corresponding theoretical and practical exams that each instructor prepares individually. We now firmly believe that learning to program is as much like learning to write: Students need to understand the intention, receive detailed feedback, rewrite and receive more advice. It is a continuous process guided by practicing in the correct contexts.

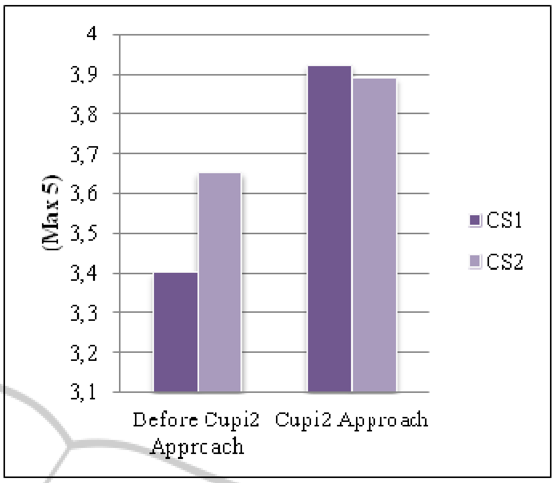


Figure 10: Average grade of students in CS1 and CS2.

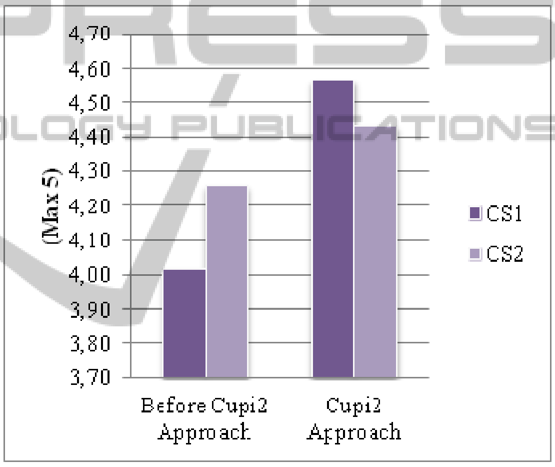


Figure 11: CS1 and CS2 quantitative evaluation by students.

Though we have obtained really promising results, the costs to support such a PbL approach are indeed high. Nevertheless, the adoption of a software factory paradigm for building these projects has greatly leveraged them. Currently, one instructor and two graduate assistants are responsible for the software factory. Approximately, they spend 30 hours in building a project for CS1 and 45 building a project for CS2. In total, 12 projects (one for each level in our courses) are fabricated in approximately 450 hours, meaning that it takes around two months to build them. Another two weeks are spent in supporting these projects since students report an average of one error per project. In conclusion, a total of 2 months and a half are spent each semester in supporting our PbL approach. These are excellent results, considering that the same people used to spend 4 months and a half building the projects before.

The time we save designing and building projects is currently invested both in scaling our approach to other universities and augmenting our courseware. On the one hand, several universities use these projects to complement their curricula in different ways. Some of them use them strictly following our approach. This means that they define similar incremental levels and, to some extent, they have fit their course syllabi to ours (sometimes even using the books we have published (Villalobos and Casallas, 2006b; Villalobos, 2008)). Others simply use the projects in the next semester we publish them. This way, they have enough time to fit them according to the learning approach they use. On the other hand, projects older than three years are published as learning objects in the Cupi2 Community (Villalobos, 2009a). Some of them are shared as projects, while others as examples. In this case, the solution of the project (the complete program) is also published. This far, our repository of projects keeps more than two hundred projects built during the six years we been using a PbL approach.

5 RELATED WORK

Project-Based Learning (PbL) is an innovative learning approach that develops several skills that are critical for success in a dynamic twenty-first century. Students are engaged in their own learning process through inquiry and they work collaboratively to create projects reflecting their knowledge. They are encouraged to "learn how to learn" via "real-life" problem solving (Adams, 2005; Laffey, Tupper, Musser, and Wedman, 1998; Nagel, 1996) inside a learning environment which several authors have considered to be more effective than traditional ones. For instance, according to Boaler (Boaler, 1999), in comparison to students at a traditional school, three times as many students from a British school following a PbL approach achieved the highest possible grade on the national exam. They concluded that students acquired a different kind of knowledge by using a PbL approach increasing their thinking skills. In another study (Bell, 2010), students in colleges from Iowa (USA) using a PbL approach raised their IOWA Test of Basic Skills scores from "well below average" to the district average in two schools and to "well above the district average" in another school. Statistics about learning assessments in these institutions grew from 15% to 90% while the district average remained the same. Similar findings in Maine (USA) concluded that a middle school using a PbL approach showed significant increases in all achievement areas on the Maine Educational Assessment Battery after only one year using the approach. The gains made by this school were three to ten times higher than the state average (Bell, 2010). In (Gallagher, Stepien, and Rosenthal, 1992), a PbL group shows an increasing problem solving ability between a pre-test and a post-test compared to the ability of a group of students following traditional learning approaches. Similarly, Schneider et al (Krajcik, Marx, and Soloway, 2002) report that PbL students score significantly higher than students nationwide on many items. Even compared with groups that traditionally score higher on achievement tests, the PbL students outscored the national average on almost half the items of the National Assessment of Educational Progress (NAEP) test. This test is the largest national representation of students' assessment known in America in various subject areas.

Students also perceive PbL to be more effective than traditional approaches. In (Curtis, 2002), students claimed that "engaging projects teaches you much more, because you get to analyze and understand the logic behind things. If you experiment and discover how things work, it will be better memorized. And if it's more fun, you'll learn faster." Similarly, high levels of satisfaction were reported by a majority of students in (Oliver, 2007), in relation to the scope of the learning, their learning success and support for their preferred learning style. Moreover, the teacher's feedback was positive and the overall learning outcomes were considered by the staff to be very satisfying.

Nevertheless, studies have shown that some students may dislike the PbL approach (Brickman, Gormally, Armstrong, and Hallar, 2002; Mohamed, 2008; Li, Dyjur, Nicolson, and Moormann, 2009). These studies report that students tend to frustrate themselves because they are engaged in activities that challenge them more than a traditional lecture session. However, they do admit they learn a lot more in a PbL learning approach. In our case, students have enhanced their perception about our courses from an average of 80% to 90% using a PbL approach.

Though PbL may seem an excellent alternative for students, its implementation involves some challenges for teachers (Lam et al, 2007). Firstly, the planning of projects implies more preparation time than lecture-based approaches. In PbL, projects need extended time periods that change between a few lessons to a whole year of education process (Köse, 2010). Secondly, some teachers may have limited or a lack of resources and support (Hall, 2002). And thirdly, teachers may resist changing their habitual pedagogical approach. For these reasons, it is critical that institutions support PbL with the appropriate methodologies and technological tools. Lam et al (Lam et al, 2007) propose that when implementing a PbL approach, institutions should support teachers for three perspectives: Competence support, autonomy support, and collegial support. In the first perspective, teachers should be engaged with PbL teaching in optimal conditions that include a good coordination, a reasonable workload and an adequate staff development. In the second perspective, teachers should be allowed to be autonomous in some parts of their teaching. They should be allowed to participate in the definition of projects, and their opinions should be acknowledged by the institution. Otherwise, they would feel threatened by the learning approach, decreasing their motivation to teach. In the last perspective, institutions should ensure a good group perception about the implementation of a PbL approach. They argue that the primary reason for some people to engage in certain behaviour is probably because this behaviour is prompted, modelled, or valued by significant others to whom they feel attached or related. To support their hypotheses, they performed a study of how school support was related to the teacher's motivation and willingness to persist in a PbL learning approach. Finally, they found that when schools are stronger in the previous perspectives, teachers show an increased motivation and willingness for implementing and continuing these PbL approaches.

Supporting PbL approaches is naturally achieved with technology. In (Köse, 2010), an advanced web system was developed to provide an effective education environment for PbL activities in a "web designing and programming" course. The web system is designed for both teachers and students. On the one hand, teachers permanently guide and control the work of students throughout the development of the tasks of a project. On the other hand, students use coding editors to accomplish these tasks, at the same time they obtain feedback from teachers and other members. The system has been assessed and the results show that 89% of the students are satisfied with the support it offers for their learning process. Teachers have reported better academic achievements as well while adopting this tool into their everyday teaching. Currently, more customization features are needed from the web system to support the PbL approach they follow in an optimal way.

6 CONCLUSIONS

In this paper we have presented our project-based learning (PbL) approach for teaching/learning computer programming and the way it is supported with a software factory for projects. We have found that after adopting such approach in our computer programming courses, namely CS1 and CS2, the average grade of students has increased, and the number of students who disapprove computer programming courses has decreased. In addition, these courses are better appreciated and students have shown high levels of motivation when working towards projects. Surprisingly, these successful indicators have not implied reducing the scope of our courses or lowering the quality of our courses. On the contrary, current courses encompass more programming concepts than before, including even concepts that derive not only from algorithmic, but also from a holistic view of computer programming.

Mainly, our PbL approach is based on incremental projects that motivate students in different ways. First, projects demonstrate them that it is not necessary to cover many topics or reach the end of the course before being able to construct something interesting. Second, projects show that programming is the balance of several domains apart from just algorithmic and programming languages. And third, projects confront students with real-world like problems from different domains. This way, students feel motivated to work since they have the feeling that the topics they are working on, whether they are simple or not, do have a real value in the course as they see how they apply them in real-world like problem solving.

One of the main drawbacks of such a PbL approach is that it is indeed highly costly for institutions. However, we have shown that an adequate software factory approach for fabricating these projects can leverage costs and generate opportunities to scale this approach to other contexts as well. In this manner, we have reduced the time we spend in the fabrication of projects by nearly a half, and several universities in Colombia have adopted our PbL approach in different ways.

ACKNOWLEDGEMENTS

We greatly acknowledge the support from European Union to our work, through the project entitled IGUAL – Innovation for Equality in Latin American Universities (code DCI-ALA/19.09.01/10/21526/ 245-315/ALFAHI (2010)123) of the ALFA III program). Although this article has been produced with the financial assistance of the European Union, the contents of this article are the sole responsibility of the authors and can under no circumstances be regarded as reflecting the position of the European Union.

REFERENCES

Adams, K., (2005) ‘The sources of innovation and creativity’, available at (accesed 10 October 2011): http://www.fpspi.org/Pdf/InnovCreativity.pdf.

Baeten, M., Kyndt, E., Struyven, K., and Dochy, F., (2010) ‘Using student-centred learning environments to stimulate deep approaches to learning: Factors encouraging or discouraging their effectiveness’, *Educational Systems Review*, vol. 5, pp 243–260, doi:10.1016/j.edurev.2010.06.001.

Bell, S., (2010) ‘Project-based learning for the 21st century: Skills for the future’, *The Clearing House*, vol. 83, pp. 39-43.

Biggers, M., Brauer, A., and Yilmaz, T., (2008) ‘Student perceptions of computer science: a retention study comparing graduating seniors with CS leavers’, *Proceedings of the 39th SIGCSE technical symposium on Computer science education*, Portland, USA.

Blumenfeld, P. C. et al., (1991) ‘Motivating project-based learning sustaining the doing, supporting the learning’, *Educational Phycologist*, vol. 26, no. 3, pp. 369–398, doi: 10.1207/s15326985ep2603&4_8.

Boaler, J., (1999) ‘Mathematics for the moment, or the millennium?’ *Education Week*, vol. 17, no. 29, pp. 30–34.

Brickman, P., Gormally, C., Armstrong, N., and Hallar, B., (2009) ‘Effects of Inquiry-based Learning on Students’ Science Literacy Skills and Confidence’, *International Journal for the Scholarship of Teaching and Learning*, vol. 3, no. 2.

Cannon, R., and Newble, D., (2000) ‘A guide to improving teaching methods: A handbook for teachers in university and colleges’, *Kogan Page*, London.

Curtis, D., (2002) ‘The power of projects’, *Educational Leadership*, vol. 60, no. 1, pp. 50-63.

Esteves , M. et al., (2009) ‘Using Second Life for problem based learning in computer science programming’, *Journal of virtual worlds research*, vol. 2, no. 1, pp.3–25.

Gallagher, S. A., Stepien, W. J., and Rosenthal, H., (1992) ‘The effects of problem-based learning on problem solving’, *Gifted Child Quarterly*, vol. 36, pp. 195-200.

Hall, S. et al. (2002) ‘Adoption of active-learning in a lecture-based engineering class’, *Proceedings of the 32nd ASEE/IEEE Frontiers in Education Conference*, Boston, USA.

Jiménez C., and Villalobos, J., (2010) ‘Design and development of an undergraduate course in Internet applications based on an integral pedagogical approach’, *Proceedings of the 2nd International Conference on Computer Supported Education (CSEDU)*, Spain.

Köse, U. ‘A web based system for project-based learning activities in ‘web design and programming’ course’, *Procedia Social and Behavioral Sciences*, vol. 2, pp. 1174-1184, doi: 10.1016/j.sbspro.2010.03.168.

Laffey, J., Tupper, T., Musser, D., and Wedman, J., (1998) ‘A computer-mediated support system for project-based learning’, *Education Technology Research and Development*, vol. 46, no. 1, pp. 73–86.

Lam, S., Wing-yi Cheng, R., and Choy, H. C., (2009) ‘School support and teacher motivation to implement project-based learning’, *Learning and Instruction*, Vol. 20, pp. 487–497, doi: 10.1016/j.learninstruc.2009.07.003.

Lea, S., Stevenson, D., and Troy, J., (2003) ‘Higher education students’ attitudes to student-centred learning: Beyond 'educational bulimia’?’, *Studies in Higher Education*, vol. 28, no. 3, pp. 321–334, doi:10.1080/03075070309293.

Li, Q., Dyjur, P., Nicolson, N., and Moormann, L., (2009) ‘Using Videoconferencing to Provide Mentorship in Inquiry-Based Urban and Rural Secondary Classrooms’, *Canadian Journal of Learning and Technology*, vol. 35, no. 3.

Mohamed, A. R., (2008) ‘Effects of Active Learning Variants on Student Performance and Learning Perceptions’, *International Journal for the Scholarship of Teaching and Learning*, vol. 2, no. 2.

Nagel, N. G., (1996) ‘Learning through real-word solving: The power of integrating teaching’, *CA Corwin Press*.

Oliver, R., (2007) ‘Exploring an inquiry-based learning approach with first-year students in a large undergraduate class’, *Innovations in Education and Teaching International*, vol. 44, no. 1, pp. 3–15.

Reinke, R. and Michalski, R., (1998) ‘Incremental learning of concept descriptions: A method and experimental results’, *J. Hayes, D. Michie, and J. Richards (Eds.), Machine Intelligence 11*, Oxford Clarendon Press.

Schneider, R. M., Krajcik, J., Marx, R. W., and Soloway, E., (2002) ‘Performance of Students in Project-Based Science Classrooms on a National Measure of Science Achievement’, *Journal of research in science thinking*, vol. 39, no. 5, pp. 410-422.

Villalobos, J., and Casallas, R., (2006) ‘Teaching/learning a first object-oriented programming course outside the CS curriculum’, *10th Workshop on Pedagogies and Tools for the Teaching and Learning of Object Oriented Concepts - ECOOP* (European Conference on Object-Oriented Programming).

Villalobos, J., Casallas, R., (2006) 'Fundamentos de Programación: Aprendizaje Activo basado en Casos', Prentice-Hall.

Villalobos, J., (2008) 'Introducción a las Estructuras de Datos: Aprendizaje Activo basado en Casos', Prentice-Hall.

Villalobos, J., Calderón, N., and Jiménez, C., (2009) 'Cupi2 community: Promoting a networking culture that supports the teaching of computer programming', *Proceedings of the 1st International Conference on Computer Supported Education (CSEDU)*, Portugal.

Villalobos, J., Calderón, N., and Jiménez, C., (2009) 'Developing Programming Skills by Using Interactive Learning Objects', *Proceedings of the 14th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE)*, France.

Woodley, M., and Kamin, S., (2007) 'Programming Studio: A course for improving programming skills in undergraduates', *Proceedings of the 38th technical symposium on computer science education*, Kentucky, USA.