International Journal of Engineering & Technology, 7(4.16) (2018) 1-4

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research paper

A Teaching Experience of Two Courses About Modeling Methods and Object-Oriented Programming in an Undergraduate Program

Freddy Paz[1]*, Freddy A. Paz[2], Juan Jesús Arenas[1]

[1]Pontificia Universidad Católica del Perú, San Miguel, Lima 32, Lima, Peru
[2]Universidad Nacional Pedro Ruiz Gallo, Lambayeque, Peru
**E-mail: fpaz@pucp.pe*

Abstract

The qualified use of advanced programming techniques for the development of enterprise software products, as well as, the proper application of modeling techniques in the design of software systems are two extremely relevant aspects in the curricula of an undergraduate software engineering student. In the Pontifical Catholic University of Peru (PUCP), these two important topics are taught by separated, in two different courses at the seventh semester of the Software Engineering career. The problematic situation with this curricular plan structure is that the students fail to relate concepts and theories whose usage is dependent one from the other. In a software development process, the analysis and design phases are key factors for the construction of a software that meets real needs of a certain group of users. However, when the computational programming and the modeling of software products are taught as independent concepts, the students tend to design only to complete the steps of the development process, and to write computer instructions without a vision and previous analysis of the system to be implemented. In this paper, we describe a teaching experience in which the authors employ collaborative strategies in a way that the students can figure out how the concepts are related and relevant for the construction and development of software products with high-quality standards.

Keywords: *university teaching experience; educational case in software engineering; collaborative teaching; software development in an academic environment; a methodology for teaching in engineering*.

1. Introduction

According to the curriculum guidelines proposed by ACM (Association for Computing Machinery) for undergraduate degree programs in Software Engineering [1], a graduate from this professional career should be able to design and implement high complex software components by applying recognized and proved principles of engineering. As a professional of this discipline, the graduate must be trained to manage all the aspects related to the traditional software development process [8], such as (1) business modeling, (2) requirements, (3) analysis & design, (4) implementation, (5) test and (6) deployment. Given the relevance of these topics, the undergraduate programs in Software Engineering usually establish a curricular plan structure which emphasizes two subjects: programming languages and modeling of software applications [3].

In the *Pontifical Catholic University of Peru* (PUCP) located in Lima – Peru, there are two mandatory courses that are taught at the seventh semester of the Software Engineering career: “Programming Languages 2” whose purpose is to train the students in advanced techniques of object-oriented programming for the development of software products, and “Information Systems 2” whose goal is the instruction in the Unified Modeling Language (UML) [5] for the correct design and analysis of the system to be implemented. Although these two topics belong to different phases of the software development process, the techniques used in each stage are highly related, since the correct accomplishment of the users’ requirements and needs would depend on the level of analysis and quality in the design that the entire team performs previously [7]. However, because of the separation of both concepts in two different courses, the students hardly notice the relationship between both. The execution of a proper analysis and design of the system is important because it provides a vision of the architecture and functionality is required to give support to the users’ needs [6].

Our methodological teaching proposal was oriented to achieve that the students can figure out the relevance of the modeling techniques for the subsequent construction of a software product. Therefore, we proposed a “collaborative teaching approach” [11] in which the professors of both courses worked together to define the contents to be discussed during each class session. The purpose was to establish a teaching structure in which the professor of a specific course could use the learned contents in the other one, as an input to explain the topics to be addressed in the current class. This schema of work was complemented with the conduction of an academic project in which the students were requested to implement a software system by the use of design and development techniques.

In this study, we describe the teaching experience and analyze the results of the collaborative strategies that were employed in our new methodological proposal for the teaching of a programming software course in collaboration with a design software system course in an undergraduate program of Software Engineering at a Latin America University.

2. Collaborative Learning

The collaborative learning is an educational approach which involves teams of students working together to solve a common problem, complete a specific task or create a particular product [4]. The members of the team are not only responsible for their own learning, but also for helping their peers to learn, causing an environment of achievement.
The didactic technique of collaborative learning involves the students in activities in which they must process information, resulting in greater retention of the subject matter, better attitudes toward learning, an improvement of the interpersonal relationships and regarding each of the members [10]. Some of the advantages of the collaborative learning are [12]:

- The technique allows students to identify the individual differences, increasing the interpersonal development.
- The technique allows students to be involved in their own learning, contributing to the achievement of a common goal and to the learning of the whole group, and giving support to teamwork.
- The group assignments and cooperative activities induce students to work by the mutual benefit in an environment in which each member recognizes that there is a common goal and that all the whole team succeeds if everyone collaborates.

In the present study, we describe the results of a collaborative learning experience in which two relevant courses of the undergraduate program in Software Engineering were redesigned to give support to an academic activity that was developed by the students during all the semester. The purpose was that the students could notice the relationship between modeling and implementing a software product. In addition, the teaching objectives are achieved because of the positive interdependence in the collaborative learning process [9]. The students become experts in the content at the same time they develop teamwork skills [2]. They begin to share goals, resources, achievements and an understanding of the relevance of each team member. A student cannot succeed independently. The positive interdependence is achieved when a member of the team perceives himself bound up with the other members, in a way that he can not succeed unless all the others do [13].

3. Case study: a Teaching Experience

The undergraduate program in Software Engineering of the *Pontifical Catholic University of Peru* (PUCP) is based on a five-year curriculum plan that the students must pass for the obtaining of the Bachelor's degree. The courses in which the collaborative teaching strategies were applied belong to the seventh semester of the professional career. The purpose of both courses is described as follows:

- **Information Systems 2:** This course is intended to develop the students' skills to analyze, design and model information systems using object-oriented technologies. At the end of the semester, the students will be able to propose and document an appropriate software architecture for a particular system that gives support to specific requirements. Finally, the course provides the students with the knowledge and tools that are required to properly analyze the quality level of a software product.
- **Programming Languages 2:** This course is intended to instruct the students in the development of object-oriented programs with graphical user interfaces. The requisites for the students to enroll in this course is to have passed a previous signature called "*Programming Languages 1*" whose content is about structured programming, algorithms and data structures in C and C++. However, in this course, the students learn advanced programming techniques to be able to apply object-oriented programming concepts in the entire process of the software development. The programming languages which are employed for this purpose are Java and C#.

3.1. The Traditional Teaching

Although "*Information Systems 2*" and "*Programming Languages 2*" are closely related courses in which the same groups of students are enrolled, there was never an attempt to propose a collaborative learning strategy. The traditional teaching was always based on the conducting of an independent academic assignment that the students had to perform in each course by separated. In the case of "*Information Systems 2*", the students are usually requested to analyze, model and design a software system without implementing it. Likewise, in the case of "*Programming Languages 2*", the students are requested to implement a system, the one that is completely different to the requested in the course of "*Information System 2*", without considering design activities whose results can guide the programming. At the end of the semester, the students accomplish two results: the software architecture of a system A and the implementation of a system B. In none of the cases, the students experiment the entire software development process. Even more, not promoting scenarios in which the design phase is complemented with the implementation phase, will cause that the students do not notice the relation between both parts of the process, neither the relevance of establishing a proper design. The traditional methodology as well as the agile approach [14] establish that the architecture of a system should guide the implementation process. The programming must be performed according to the design diagrams that have been developed. However, given that there is not a connection between both concepts, the students design software systems without an understanding that this specific artifact is an input for the development process.

3.2. Description of the Teaching Proposal

The new teaching proposal is illustrated in Figure 1. We reorganize the content of both courses in a way that the subjects that were taught in the week for one course, serve as an input for the other course. The proposed structure is shown in Table I. For instance, with knowledge of the main techniques that are employed to identify the requirements of a system (Week 2 – *Information Systems 2*), the students would be prepared to learn how to abstract and model the reality in *objects* (Week 2 – *Programming Languages 2*). In this session, the students would learn the meaning of *classes* and *methods*, as well as the programming techniques that are related to these concepts. Correspondingly, the knowledge about *class*, *attribute* and *method* would allow the students to diagram *uses cases* (Week 3 – *Information Systems 2*). The teaching of modeling techniques would give support for following subjects in the fourth week of *Programming Languages 2* associated to the topics of *inheritance* and *polymorphism.* The academic semester in PUCP is composed of fifteen weeks of which two of them (Week 7 and Week 15) are oriented to evaluate through exams the knowledge of the students. Considering this aspect, we elaborated two syllabuses in which the contents of one course complement to the other one.

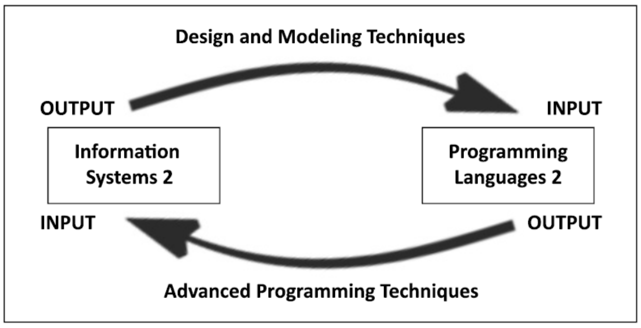

Fig. 1: Educational methodology proposed for the teaching of both courses

In addition to the new pedagogical structure of the courses, we also modified the evaluation methodology to incorporate a collaborative activity in which the students can apply the learned concepts in the development of a real case study. They were requested to design and implement a software product by following all the modeling guidelines and using the programming techniques that were taught during the class sessions. Given that the same group of students was enrolled in both courses, it was not a problem to divide the class into small teams. The twenty-two students were divided into three teams of four students, and two teams of five students. The type of software to be developed by each team was left to the election of themselves. The had to be developed gradually according to the topics that were discussed until the moment in class. In all the even weeks, the students had to present the progress of their projects during special lab sessions as is shown in Figure 2. During these sessions, the teams received feedback about their design and implementation decisions. The purpose was that all students learn from their own mistakes and finish both courses with the development of a high-quality software product, whose construction process and final results would be presented at the week 14.

Table 1: Content of each week by course

Week number	Information Systems 2	Programming Languages 2
Week 1	Introduction to Unified Modeling Language	Introduction to Object Oriented Programming
Week 2	Requirements engineering	Objects, classes and methods
Week 3	Use cases diagramming and database design	Inheritance, abstraction and polymorphism
Week 4	System analysis and class diagram design	Namespaces, libraries and packages
Week 5	Effort estimation	Persistence and file handling
Week 6	Software architecture and component diagram design	Lists, arrays, events, exceptions and graphics programming
Week 7	Mid-term evaluation	
Week 8	Activity diagram design and interaction diagram design	Windows and graphical user interfaces design
Week 9	Deployment diagram design and Design patterns – Part 1	Multitier architecture development
Week 10	Design patterns – Part 2	Database connection
Week 11	Usability engineering	Concurrent programming
Week 12	Software testing	Distributed programming
Week 13	Software quality	Web services and reports
Week 14	Presentation of the final project	
Week 15	Final evaluation	

Fig. 2: Example of a special lab session

4. Analysis of the Results

The teaching proposal was applied during the first semester of 2017. The professors of both courses made efforts to be present during all the special lab sessions in order to offer proper feedback to each one of the teams. There was a wide diversity of type of software applications that were developed by the students. The only requisite we established as teachers was the use of *Unified Modeling Language* (UML) and *Object-Oriented Programming* (OOP) for the development of these systems. A brief description of the students' proposals is presented as follows:

- **A Carpooling System:** Considering that Lima, the capital of Peru, is a big city where the distances are long and the transportation from home to the university can take up to two hours, this team proposed the development of a carpooling system for exclusive use of the university community. Carpooling is the sharing of car journeys, in a way that if someone near your home is driving to the university, you can request this person to help you in the transportation.
- **A Web System for Question/Answers about Online Classes:** This Web system was developed for educational institutions that offer online courses. The purpose of this application is to serve as a way of communication between the institution and the people who are interested in the offered classes. The system provides a platform in which the users can register a question and the institution give answers.
- **A Video-Game:** Given the current success of Pokemon games during the last years, the students of this team decided to develop an RPG video-game based on this franchise. In this adventure, the player walks by different maps in an intense search of several wild creatures called Pokemon. The game implements battles with NPC who are controlled by artificial intelligence.
- **An Integrated University Library System:** Following the constraints and rules of the libraries in our university, one of the teams proposed the development of an integrated system. At present, each library works with an independent software for the search and borrowing of books. The purpose of this system was to provide with a software tool in which the university community could access the information of all the libraries that are located on our campus.
- **Gym Management Software:** In the last year, people are given more importance to the regular physical activity because of the health benefits. In this sense, one of the teams decided to implement a platform to give support to the main activities that take place in a gym. The purpose of this system was to manage customers' memberships of a gym, coaching, sale of fitness products and registration to classes offered by the body training center.

All the systems were developed using Java and C#, with a MySQL server connection. The purpose of the new educational methodology was achieved given that the students begin with the design of the system and could notice the relevance of this phase at the time of the programming. The students figured out that their program must be developed according to the architecture they have previously defined. In this sense, many changes were proposed during the software process development and both the design and the programming had to be fixed together.

Regarding the collaborative activities, the professors could observe during the special lab sessions that the team members helped each other in an efficiency and effective way. The teams promoted the success of all their members, by exchanging information and knowledge between them. We also highlight the way in which the team members offered feedback to improve the performance of the others, how they analyze the results as a team, and finally the process they followed to make reflexions in order to achieve better quality results. We could notice that the collaborative work encourages cognitive activities and interpersonal dynamics. For instance, when the students promote learning at their partners, when a student explain to other how to program or design certain functionality of the system, or when the students discuss the concepts.

5. Conclusions and Future Works

Two relevant topics in the curricula of the undergraduate program in Software Engineering are the lessons about the use of advanced programming techniques as well as the instruction in modeling methods to design appropriate software architectures. Both aspects represent important phases of the software development process. In the *Pontifical Catholic University of Peru*, these topics, despite being widely related, are taught by separated in two different courses: *Programming Languages 2* and *Information Systems 2*. However, in this way, the students used to program without having a vision of the software design, that according to the theory should be a prerequisite to the implementation phase. In the same way, the students used to propose a software architecture without making it tangible in a development process. Therefore, the professors of both courses established a new pedagogical approach to improve the learning process.

The teaching proposal involved a re-organization of the contents of both courses in a way in which in all weeks, the lessons provided in one of the courses serve as input for the understanding of the concepts to be discussed in the other course. Moreover, the fact that the same group of students was enrolled in both courses allow us to promote a single project in which the students were requested to work collaboratively for the development of a software product. The application of the methodology took place during the first half of 2017 and demonstrates being effective and efficient. The students could notice the relevance of the design decisions, and how the architecture design can affect the programming of a software product. The collaborative strategies allow the team members to learn from their partners and to increase their cognitive and interpersonal abilities. The students recognize that the success is only possible if all members work together for the common goal. Although the proposed methodology offers effective results, it is still necessary to perform more case studies in order to generalize the results for different academic contexts. The results can be complemented with future interviews and questionnaires in order to determine the students' perception about the academic assignment they had to perform.

References

[1] ACM, Software engineering 2014 – curriculum guidelines for undergraduate degree programs in software engineering, Association for Computing Machinery, 2015.

[2] Agarwal A, Seretse OM, Letsatsi MT, Marumo R & Mokgwathi T, "Role of academia-industry collaboration in enriching engineering education: A case study in sub-sahara context", International Journal of Engineering & Technology, vol. 7, no. 3, 2018, pp. 1360-1365.

[3] Alarifi A, Zarour M, Alomar N, Alshaikh Z & Alsaleh M, "SECDEP: Software engineering curricula development and evaluation process using SWEBOK", Information and Software Technology, vol. 74, 2016, pp. 114-126.

[4] Barkley EF, Cross KP & Major CH, Collaborative learning techniques: A handbook for college faculty, 1st ed., San Francisco, CA, USA: Jossey-Bass Higher and Adult Education, 2004.

[5] Booch G, Rumbaugh J & Jacobson I, The Unified Modeling Language User Guide, 2nd ed., Reading, MA, USA: Addison-Wesley Professional, 2005.

[6] Gorschek T, Tempero E & Angelis L, "On the use of software design models in software development practice: An empirical investigation", Journal of Systems and Software, vol. 95, 2014, pp. 176-193.

[7] Haraty RA & Hu G, "Software process models: A review and analysis", International Journal of Engineering & Technology, vol. 7, no. 2.28, 2018, pp. 325-331.

[8] Jacobson I, Booch G & Rumbaugh J, The Unified Software Development Process, 1st ed., Reading, MA, USA: Addison-Wesley Professional, 1999.

[9] Laal M, "Positive interdependence in collaborative learning", Procedia – Social and Behavioral Sciences, vol. 93, 2013, pp. 1433-1437.

[10] Laal M & Ghodsi SM, "Benefits of collaborative learning", Procedia – Social and Behavioral Sciences, vol. 31, 2012, pp. 486-490.

[11] Laal M, Khattami-Kermanshahi Z & Laal M, "Teaching and education; collaborative style", Procedia – Social and Behavioral Sciences, vol. 116, 2014, pp. 4057-4061.

[12] López-Yáñez I, Yáñez-Márquez C, Camacho-Nieto O, Aldape-Pérez M & Argüelles-Cruz AJ, "Collaborative learning in posgraduate level courses", Computers in Human Behavior, vol. 51, 2015, pp. 938-944.

[13] Nam CW & Zellner RD, "The relative effect of positive interdependence and group processing on student achievement and attitude in online cooperative learning", Computers & Education, vol. 56, no. 3, 2011, pp. 680-688.

[14] Papadopoulos G, "Moving from traditional to agile software development methodology also on large, distributed projects", Procedia – Social and Behavioral Sciences, vol. 175, 2015, pp. 455-463.