ADPT: AN ACTIVE LEARNING METHOD FOR A PROGRAMMING LAB COURSE

Claudia Martínez, Marcia Muñoz
Departamento Ingeniería Informática
Universidad Católica de la Santísima Concepción, Chile
cmartinez@ucsc.cl, marciam@ucsc.cl

ABSTRACT

This paper describes the active learning method used in a programming lab course in the Computer Science program at the Universidad Católica de la Santísima Concepción (UCSC). As a result of the UCSC School of Engineering curricular reform performed in 2011, the first year of the computer science program was modified to include a Programming Lab course where teams of students analyze problems and design solutions following a structured approach. Each stage of this process is supported by specific tools and techniques.

ADPT (Analysis, Design, Programming and Testing) is an active learning method based on a PBL (Problem-Based Learning) approach, composed of the four stages of the classical software development method, also called the waterfall model. This approach includes rigorously ordered stages, where each one assumes the previous stage has been completed. The main difference between ADPT and PBL is that PBL encourages collaboration within the working team whereas ADPT also encourages collaboration with other teams. In the Programming Lab course, students have to solve four problems using the ADPT method.

The results discussed in this study correspond to students that took the Programming Lab course in the first semester of 2013. Preliminary results are positive both in relation to the assessment of the results obtained by the student teams as well as students' perception of the ADPT method. Surveys also registered high levels of satisfaction with active learning methods, especially team-work and cooperative learning. These results are consistent with our previous work in 2011 and 2012.

KEYWORDS

Active learning, problem-based learning, cooperative learning, standards: 5, 7, 8 and 11.

INTRODUCTION

In 2011, the UCSC School of Engineering reformed the curricula of its five engineering programs based on the CDIO initiative (Loyer et al., 2011). Among other issues, the curriculum reform process addresses the problem of motivating its first years' students by incorporating first-year courses (Muñoz et al., 2013) through the development of activities that acquaint students with their professional role, thus contributing to the adoption of CDIO standards 1, 4 and 8. In particular, the computer science program was modified to include two semester-length introductory courses. In the first course, Introduction to Computer

Science, students become familiarized with their profession and with the software lifecycle by developing a simple project from its conception to its operation. The second course is a Programming Lab where students analyze problems and design solutions following a structured approach. In this course, students engage in programming and also develop personal skills for self-learning and teamwork.

FRAMEWORK

Problem Based Learning (PBL)

PBL is active learning method in which teams of students learn through solving relevant problems and reflecting on their experiences (Barrows & Tamblyn, 1980). Problems must involve a cognitive conflict, and must be challenging and motivate students to seek a solution. Problems must be complex, so that its solution requires cooperation among all team members. The instructor must act as a facilitator, making the problem a real team challenge to be solved and thus preventing students from just dividing up the work. In this way, PBL not only helps students learn the subject matter, but also helps develop teamwork, self-directed learning, information searching from diverse sources, decision making, problem solving in different ways oral and written communications, among others (Prince, 2004).

Cooperative Learning

Cooperative learning is the instructional use of small teams so that students work together to maximize their own and each other's learning (Johnson, Johnson & Smith, 1998). Felder and Brent (1994, 2007) discuss the elements of cooperative learning, such as:

Positive interdependence: Team members must rely on one another to achieve the goal. If any team members fail to do their part, everyone suffers the consequences.

Individual accountability: All students in a team are held accountable for doing their share of the work and for their mastery of all of the material to be learned.

Face-to-face promotive interaction: Although some of the team work may be divided and done individually, some must be done interactively, so that group members provide one another with feedback, challenge each other's conclusions and reasoning, and perhaps most importantly, teach and encourage each other.

Appropriate use of collaborative skills: Students are encouraged and helped to develop and practice trust building, leadership, decision-making, communication, and conflict management skills.

Team processing: Team members must set goals, periodically assess what they are doing well as a team, and identify the changes to be made to function more effectively in the future.

THE ADPT METHOD

ADPT (Analysis, Design, Programming and Testing) is an active learning method based on a PBL approach and cooperative learning (Figure 1), composed of the four phases of the

classical software development method (waterfall model) (Pressman, 2009). This approach includes rigorously ordered stages, where each stage assumes the previous stage has been completed.

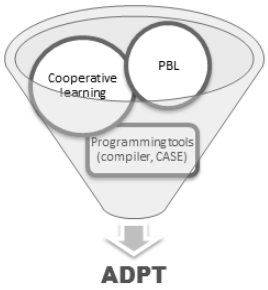


Figure 1. ADPT sources

The instructor's role as a facilitator consists of rigorously planning the activity and of guiding students along the different stages of the method. For a given problem, students must identify inputs, processes, outputs and restrictions. Then, they must design, build and test a solution by following the stages described below:

Induction

The instructor presents an example animation, which was built as a learning object in Macromedia Flash. The instructor formulates key questions, which are discussed in class, and clarifies students' questions about the problem. 🕓: 10 minutes.

Division into Teams

Students are divided into N teams, where N is a multiple of 4. Ideally, each team should have 3 or 4 members. Team size must be adjusted so that the number of teams is always a multiple of 4. Teams are labeled $T_1..T_N$. 🕓: 5 minutes.

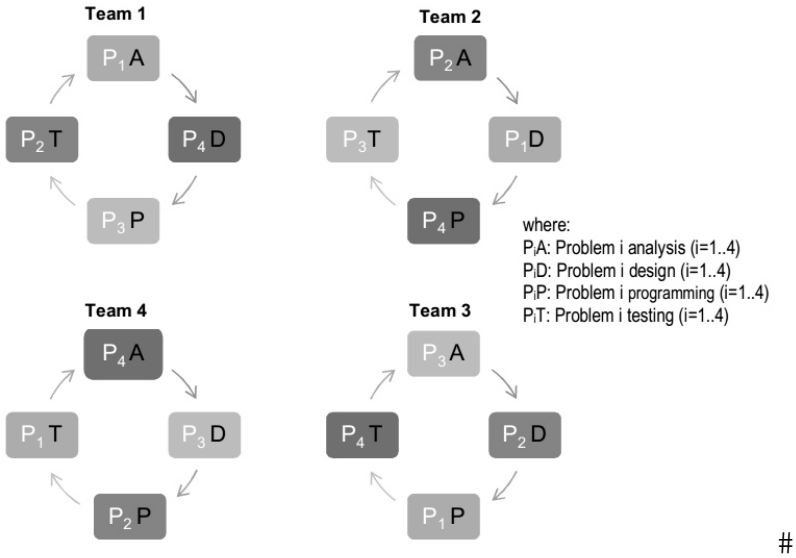
The instructor now assigns a list of 4 problems Pj to each team (j=1..4), explains the didactic sequence execution, and hands out other materials and the deliverables' format. 🕓: 5 minutes.

Didactic Sequence

Every team (T_i, i=1, 2,..N) executes the sequence of problem resolution tasks shown in Figure 2. In this sequence, each team performs all four stages of the classical software development method (waterfall model): analysis, design, programming and testing for a different problem, so that one team's deliverable is the next team's input. For example, $Team_1$ begins by analyzing problem P_1. Its results follow the analysis template shown in Figure 3, and must be clear and precise. $Team_2$ uses this deliverable as an input to the design stage, which generates as output a flowchart. $Team_3$ uses this flowchart as an input to the programming stage. Finally, $Team_4$ tests the program generated by $Team_3$. 🕓: 60 minutes.

Closing Stage

The instructor finalizes the activity, analyzing the process, student learning and team results, remarking on difficulties in generating good specifications, controlling analysis time, balancing workload among teammates, etc. 🕓: 10 minutes.



#

Figure 2: Teams sequence

This didactic sequence has an estimated overall time of 90 minutes. Table 1 shows the ADPT sequence of actions for each team.

Table 1. ADPT sequence for each team

Team	ADPT sequence
T_1	$P_1A \rightarrow P_4D \rightarrow P_3P \rightarrow P_2T$
T_2	$P_2A \rightarrow P_1D \rightarrow P_4P \rightarrow P_3T$
T_3	$P_3A \rightarrow P_2D \rightarrow P_1P \rightarrow P_4T$
T_4	$P_4A \rightarrow P_3D \rightarrow P_2P \rightarrow P_1T$

Figure 3 shows the deliverables for each stage. The analysis stage delivers a completed analysis template, the design stage delivers a flowchart, the programming stage delivers source code, and the testing stage delivers the results of executing the test cases.

Comparison to other Active Learning Methods.

The ADPT method is a systematization of regular classroom practices from the Programming Lab I course. Even though the ADPT method was proposed to support teaching computer programming, it includes generic elements of both PBL and cooperative learning methods, and adds techniques and tools from the computer programming domain, as shown in Table 2. It can be seen from the table that the main differences between ADPT and PBL are that PBL encourages collaboration within the working team whereas ADPT also encourages collaboration with other teams, and that PBL focuses on solving ill-structured relevant problems, while ADPT focuses on well-structured and semi well-structured simple problems.

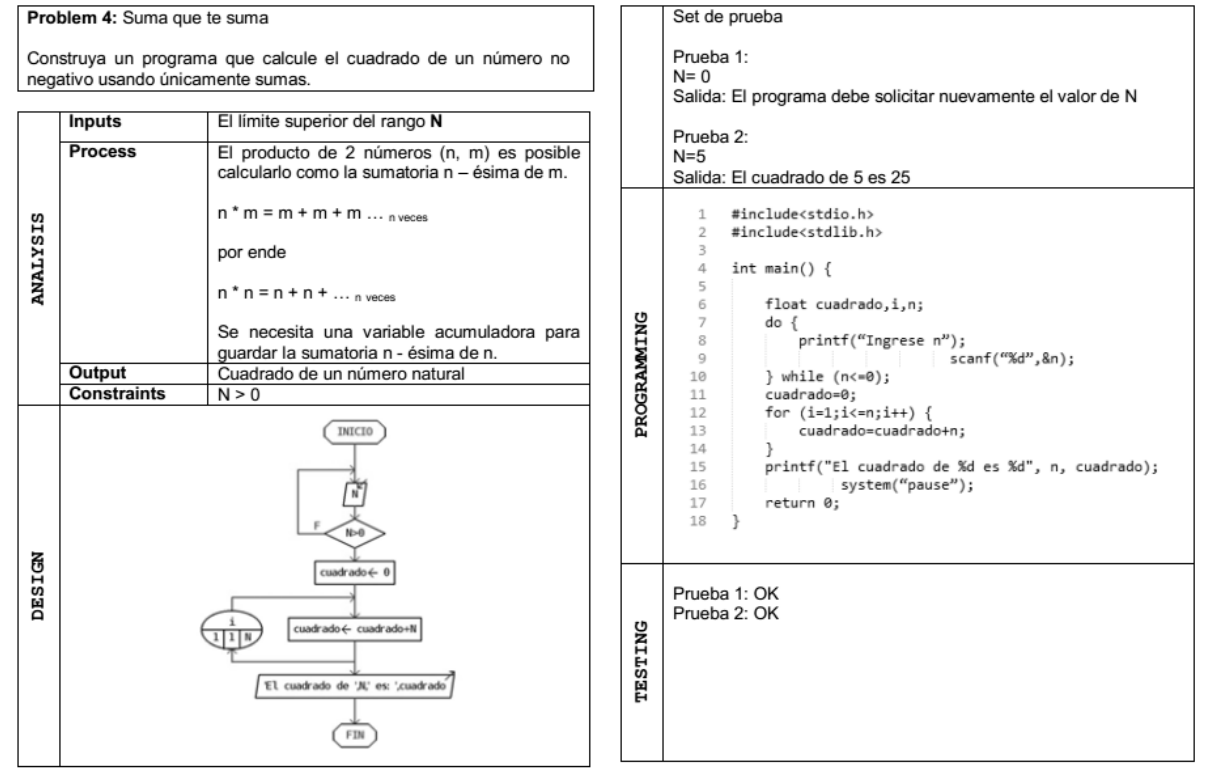


Figure 3. ADPT deliverables for each stage

Table 2. Comparison of active learning methods

	CDIO		Problems types	Teacher role as facilitator	Interaction	Positive Interdependence	Specific domain	Use rubrics	Group size
	goals	standards							
PBL	2.1	7, 8, 11	ill-structured problems	X	Face to face inside group	X	None	X	Small
Cooperative Learning	2.1	7, 8, 11	Any	X	Face to face inside group	X	None	X	Small
ADPT	2.1 3.1	5, 7, 8, 11	Structured and semi-structured	X	Face to face, inside and between groups	X	Teaching Programming	X	Small

Adoption of the ADPT method contributes to CDIO syllabus goals 2.1 and 3.1, as well as to CDIO standards 5, 7, 8 and 11. Standard 5 refers to design and implementation experiences, in this case at a basic level. Standard 7 refers to integrated learning experiences that foster the learning of disciplinary knowledge as well as personal and interpersonal skills, while standard 8 discusses teaching and learning based on experiential active learning methods. Finally, standard 11 covers the assessment of student learning in personal and interpersonal skills, and product, process and system building skills, as well as in disciplinary knowledge.

ADPT Method Application

Programming Lab I Course Description

In this course, students learn computer programming using basic tools and simple structured problems, and also develop teamwork skills. Table 3 presents the course's learning outcomes, associated to disciplinary knowledge and reasoning (CDIO 1.2), personal and professional skills and attributes (CDIO 2.1), and interpersonal skills: teamwork (CDIO 3.1):

Table 3. Programming Lab I contribution to CDIO syllabus goals

Programming Lab I Learning Outcomes		
Disciplinary knowledge and reasoning	1.2 Fundamentals of algorithms, data structures and programming languages	
	1.2.1	Explain the different software development stages.
	1.2.2	Identify inputs, outputs and constraints for a given problem.
	1.2.3	Design a structured solution using an algorithmic representation technique.
	1.2.4	Build an algorithmic solution using a structured programming language.
Personal and professional skills and attributes	2.1 Analyze a problem by dividing it into identifiable parts, and propose solutions.	
Interpersonal skills: Teamwork and communication	3.1 Can work autonomously and join interdisciplinary teams.	

This course meets for 5 hours per week. During the second semester of 2013, the Programming Lab I course was taught in two parallel sessions of 24 and 29 students, respectively. Table 4 describes example problems belonging to each of the three applied didactic sequences on a semester.

Table 4. ADPT example problems

ADPT problem	Short Description
ADPT1: Piano melodies	Develop an application using the ADPT method that simulates an 8-key piano (do-re-mi-...sol), where each key must reproduce a different sound. The program must terminate when the user presses the * key.
ADPT2: Car crashes	Develop an application using the ADPT method that simulates a moving vehicle using ASCII characters in the C programming language. The vehicle must move to the right with a 1 second lag. Parameterize the initial position and speed (consider it constant). Some of the challenges that students had to face include parameterizing the vehicle's final position, changing the trajectory from linear to sinusoidal, and having the vehicle crash once the trip is complete.
ADPT3: Text processing	Develop an application using the ADPT method that implements a text preprocessing module for a spanish language text read from an input text file. This preprocessing must identify the number of consonants, vowels and words in the text, the number of vowels in the text, the number of words in the text, and identify stopwords present in the text.

ADPT PRODUCT- PROCESS RUBRIC

	EXCELLENT (5 points)	GOOD (4 points)	SATISFACTORY (3 points)	POOR (2 points)	UNACCEPTABLE (1 point)
A	Identifies all the inputs, outputs and constraints of the problem.	Identifies most of the inputs, outputs, and constraints of the problem.	Identifies some of the inputs, outputs, and constraints of the problem.	Identifies a minimum amount of the inputs, outputs, and constraints of the problem.	Fails to identify the inputs, outputs and constraints of the problem.
D	Correctly designs the algorithm in pseudocode as well as test data.	Correctly designs the algorithm in pseudocode making most of the time proper use of control structures, and correctly design test data.	Designs the algorithm in pseudocode and does not design test data.	Designs the algorithm poorly, making improper use of control structures and does not design test data.	Fails to design either the algorithm or the test data.
P	Codes and compiles the algorithm in the Pseint tool and also generates its flowchart. The student is able to solve syntactic and semantic errors without help.	Codes and compiles the algorithm in the Pseint tool. The student is able to solve syntactic and semantic errors with little help.	Codes and compiles the algorithm in the Pseint tool. The student is able to solve syntactic and semantic errors with considerable instructor help.	Represents the algorithm only in pseudocode using the Pseint tool, and is unable to solve syntactic and semantic errors even with instructor help.	Fails to code the algorithm.
T	Code passes more than 90% of test data.	Code passes 71 % to 90% of test data.	Code passes 50% to 70% of test data.	Code passes less than 50% of test data.	Code fails all test data.

Figure 4. Product-process rubric

ADPT TEAMWORK RUBRIC

	EXCELLENT (5 points)	GOOD (4 points)	SATISFACTORY (3 points)	POOR (2 points)	UNACCEPTABLE (1 point)
Attitude (10%)	Is never publicly critical of the work of other team members. Always has a positive attitude about the task.	Is rarely publicly critical of the work of other team members. Always has a positive attitude about the task.	Is occasionally publicly critical of the work of other team members. Occasionally has a negative attitude about the task.	Is often publicly critical of the work of other team members. Often has a negative attitude about the task.	Is often publicly critical of the work of other team members. Always has a negative attitude about the task.
Development (10%)	Prepares and plans the work to be done in collaboration with other team members. Supports others in making outlines, diagrams, and tables.	Develops and supports others in making outlines, diagrams, and tables.	Sometimes supports others in making outlines, diagrams, and tables.	Rarely supports others in making outlines, diagrams, and tables.	Never develops and supports others in making outlines, diagrams, and tables.
Work quality (20%)	Generates work of the highest quality.	Generates work of high quality.	Generates work that occasionally requires review by other team members to ensure quality.	Generates work that often requires review by other team members to ensure quality.	Generates bad quality work that always requires review by other team members to ensure quality.
Problem-solving (40%)	Looks for and suggests solutions to problems.	Refines solutions suggested by others.	Occasionally suggests or refines solutions to problems.	Does not suggest or refine solutions, but is willing to try out solutions suggested by other team members.	Does not try to solve problems nor makes contributions to problem solving.
Working with others (20%)	Always listens to, shares with, and supports the efforts of others. Tries to keep team members working well together.	Usually shares with, and supports the efforts of others. Tries to keep team members working well together.	Occasionally shares with, and supports the efforts of others.	Rarely listens to, shares with, or supports the efforts of others.	Never listens to, shares with, or supports the efforts of others.

Figure 5. Teamwork rubric

ADPT didactic sequence qualification

The UCSC grading system establishes a numeric scale (1-7). Therefore, the points gathered from the rubrics must be transformed to be able to calculate each activity's final grade for each student, considering the process-product evaluation as 80% of the grade and the teamwork evaluation and 20% of the grade.

Reflective Memo ADPT Sequence

This instrument is anonymous and gives us your perception about the activity that you have done recently with the ADPT method.

A. About the ADPT sequence

	Highly suitable	Suitable	Neutral	Unsuitable	Highly unsuitable
Evaluate the activity in relation to the achievement of the stated goals					

B. Indicate the achievements and benefits of having developed the activity

C. Indicate the difficulties in implementing the activity

Figure 6. Reflective memo template

RESULTS AND DISCUSSION

Figure 7 shows the results obtained from section A of the reflective memo shown in Figure 6.

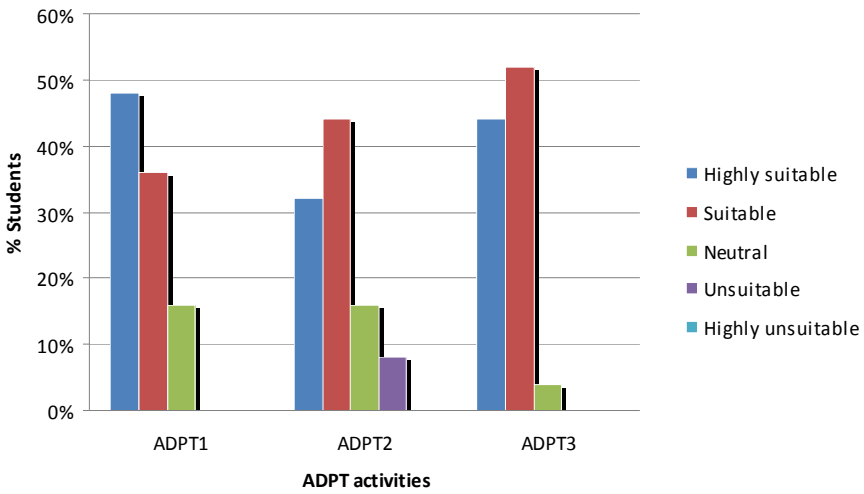


Figure 7. Pertinence of each ADPT didactic sequence

In the case of the ADPT1 activity, 84% of students consider it suitable or highly suitable for achieving the learning outcomes. In the case of the ADPT2 and ADPT3 activities, the corresponding results are 76% and 96%, respectively.

Sections B and C of the reflective memo shown in Figure 6 allowed gathering student comments about the ADPT sequences. Among the positive comments received, we find comments such as "*activity ADPT1 was fun, it made us work as a team, and I could contribute my ideas for solving the problem*", "*for ADPT2, we found out how to use mathematical functions and how to use external libraries*", "*we could associate the vehicle's movement to a mathematical function such as sin(x), cos(x), it was great!*", "*in ADPT3, we*

learned how to use the string library on a real text processing problem". Also, negative comments such as "*programming is hard for me so I needed more time to finish the activity*", or "*I didn't know all the control structures needed for this activity*" give us the chance to detect problems and study avenues for future activity improvements.

Table 5 shows the evolution of individual grades related to the activities of the three ADPT didactic sequences (conditional structures, iterative structures and string handling) for the semesters II-2011, II-2012 and II-2013. In 2011 and 2012, these topics were each evaluated using a programming test, whereas in 2013 these topics were evaluated using the ADPT method. Even though student cohorts are generally homogeneous from one year to the next, the II-2013 semester grades improvement cannot be solely attributed to the ADPT method. However, high student motivation and positive student comments about the method lead us to believe that the ADPT method is a positive didactic improvement in teaching programming.

Table 5. Evolution of grades

	II- 2011			II-2012			II -2013		
	Test 1	Test 2	Test 3	Test 1	Test 2	Test 3	ADPT1	ADPT2	ADPT3
% Passing grades	15%	38%	19%	30%	35%	40%	80%	85%	72%
Average (1 to 7 scale)	2,8	4,2	3	3,5	4,4	4,6	5,9	5,5	5,2

CONCLUSIONS

We have found the ADPT method to be a worthy contribution to CDIO standard 8, as it helps improve the application of active learning methods in the UCSC School of Engineering, which is currently at level 3 (Martínez et al., 2013). ADPT also contributes to the CDIO syllabus goals stated in the course: teamwork, analytical reasoning and problem solving. Our preliminary results for the ADPT method lead us to believe that it can aid the teaching of computer programming, as it improves student motivation, it helps generate valuable interactions within student teams, and also it promotes collaboration with other teams, thus contributing to CDIO standard 7 and 11 by integrating different assessment methods. The results were positive both in relation to the assessment of the results obtained by the student teams as well as students' perception of the ADPT method. Student comments also show that some students have trouble elucidating unknown concepts by searching for relevant information, even in the case of well-structured or semi well-structured problems. This weakness may be addressed the following semester by having the instructor intervene directly and identify common conceptual weaknesses in each group, and explaining them either in situ, or in a previous lecture, to improve the students' input behavior. On the other hand, phrases such as "*even though problems were a bit complicated because we didn't know all the control structures, we shared ideas, discussed them and solved the problem*" show that students recognize the value of teamwork and of learning in a real context. Even though ADPT is an introductory-level design implementation experience contributing to CDIO standard 5, it can be exported and adapted to higher-level courses such as Software Engineering Workshop, which is currently taught using a Project-Based Learning approach.

The School of Engineering at UCSC is working on systematizing the application of student-centered methodologies. Thus, Faculty are convinced of the importance of developing both disciplinary knowledge and across-the-curriculum skills such as teamwork, critical thinking, decision making, etc., in our students, so as to give them an integral professional education.

REFERENCES

Barrows, H. S. & Tamblyn, R. (1980). Problem-Based Learning: An approach to Medical Education. New York: Springer.

Felder, R. & Brent, R. (1994). Cooperative Learning in Technical Courses: Procedures, Pitfalls, and Payoffs. Retrieved from http://www.ncsu.edu/felder-public/Papers/Coopreport.html.

Felder, R. & Brent, R. (2007). Cooperative Learning. Chapter 4 of P. A. Mabrouk, (Ed.), Active Learning: Models from the Analytical Sciences, ACS Symposium Series 970. Washington, DC: American Chemical Society.

Johnson, D. W., Johnson, R. T. & Smith, K. A. (1998). Active Learning: Cooperation in the College Classroom, (2nd ed.); Interaction Book: Edina, MN.

Loyer, S., Muñoz, M., Cardenas, C., Martínez, C., Cepeda, M. & Faúndez, V. (2011). A CDIO Approach to Curriculum Design of five Engineering Programs at UCSC. *Proceedings of the 7th International CDIO Conference*, Technical University of Denmark, Copenhagen.

Martínez C., Muñoz M., Cárdenas, C. & Cepeda, M. (2013). Adopción de la Iniciativa CDIO en los Planes de Estudio de las Carreras de la Facultad de Ingeniería de la UCSC. *Proceedings of the 11th Latin American and Caribbean Conference for Engineering and Technology.*

Muñoz M., Martínez, C., Cárdenas, C. & Cepeda, M. (2013). Active learning in first-year engineering courses at Universidad Católica de la Santísima Concepción. *Australasian Journal of Engineering Education*,19(1), 27-38.

Pressman, R. (2009). *Software Engineering: A Practitioner's Approach*, *7th.* Ed. McGraw-Hill, New York.

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

BIOGRAPHICAL INFORMATION

Claudia Martínez A. is a faculty member in the Computer Science department at UCSC, where she also serves as department head. Her research and interest areas are related to information retrieval, semantic web, programming and engineering education.

Marcia Muñoz V. is a faculty member in the Computer Science department UCSC, where she also serves as undergraduate program director. Her research interests are software engineering, artificial intelligence, machine learning, and engineering education.

Corresponding author

Mg. Claudia Martínez A.
Computer Science Department
Universidad Católica de la Santísima Concepción
Alonso de Ribera 2850, Casilla 2850
Concepción, Chile 4090541
56-41-234-5332
cmartinez@ucsc.cl