2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering Education and Training Track

What can Students Get from a Software Engineering Capstone Course?

María Cecilia Bastarrica
Computer Science Department
Universidad de Chile
Santiago, Chile
cecilia@dcc.uchile.cl

Daniel Perovich
Computer Science Department
Universidad de Chile
Santiago, Chile
dperovic@dcc.uchile.cl

Maíra Marques Samary
Computer Science Department
Universidad de Chile
Santiago, Chile
mairamarques@hotmail.com

***Abstract*—For the last ten years we have been teaching a capstone course for fifth year students of the Computer Science Department of the Universidad de Chile. Five year ago we redesigned the course, shifting from projects following a waterfall process and focused on technical aspects, to one centered in soft skills following agile practices. Since then, we provide out students a concrete learning outcome: to internalize how relevant is having and developing critical soft skills to succeed in projects. Last year, we wondered whether our students were actually getting what we declared. We conducted a survey on students' initial and final perception about the relative value and difficulty of different dimensions involved in their projects: technical challenge, teamwork, planning, and negotiation with the client. Also, we applied a one-tailed dependent pair sample t-test to determine the statistical significance of the surveys result. We found out that the relative value of soft skills grows while that of the technical challenge drops, and that the students find that planning and teamwork are harder than they expected. Also, we found statistically significant evidence that, for the soft skills we have measured, the perceived relative relevance actually changes throughout the course.**

***Index Terms*—software engineering education, computer science education, capstone course**

I. Introduction

Several authors report that recently graduated computer science professionals are not well prepared for facing industrial work [4], [7], [15]. Universities have tried to deal with this issue applying Project Based Learning (PBL) strategies [11] mainly through capstone courses [2], whose main goal is to make students work in teams to put into practice all the knowledge they have acquired during their career. One of the main characteristics of software engineering capstone courses is that students should deal with real problems similar to those they will face in industry while still counting on some guidance on how to address this reality that they are not used to. Capstone courses are normally one of the last ones in the curriculum and they demand a lot of work from students. The use of capstone courses in software engineering education is not new [6], but since the publication of SE2004 [13], they are gaining more relevance. SE104 [14] has reinforced this idea.

For the last ten years we have taught *Software Project*, a capstone course for fifth year students of Computer Science Engineering at the Universidad de Chile. Students have to work in a real world project with a real client, external to the university, during 15 weeks, devoting 16 hours a week of effective work in the client's organization.

Until five years ago students had to follow a waterfall process, loaded with documentation. However, this was not only boring for students, but also inefficient since this documentation was not strictly necessary for all clients. Since then, we started introducing some agile practices. Three of the most relevant of these practices are incremental development, timeboxing and client on site. Timeboxing allowed us to limit the amount of time spent by students in working for their projects to 16 hours per week per student. Therefore, all the work is done in this time: software development, documentation, validation, course presentations, etc. In this way, clients are more careful when asking for documentation since this would mean less software. Also, having the team work together with the client made communication easier and less dependent on documents.

The results have been encouraging: projects are almost always successfully deployed and operational at the end of the semester, and both, students and clients, are usually much happier with the results. These results are consistent with other reports on capstone courses on software engineering [5], [18]. However, we still wondered what made the difference, and what specifically the students learned during the course. Therefore, we formulated the following research question:

RQ What can students get from a software engineering capstone course?

To this end, we designed a survey that consisted on asking each team to grade the initial and final perceived value and difficulty of four dimensions: technical challenge, negotiation with the client, project planning and teamwork. These values were expressed as percentages, i.e., all item values add up to 100. Teams provided the initial values just after reading the project proposal and having a first meeting with the client. Final values were provided after the end of the course. The goal was to evaluate the variation of these two measurements, i.e., what students have learned during the course.

We found that the perceived relative value of correctly addressing technical challenge dropped significantly after the course. Therefore, they realized that soft skills were much more determinant for the success of the project. We also found that all the analyzed dimensions had a significant change in their initial and final perception, either in value or difficulty, and

978-1-5386-2671-9/17 $31.00 © 2017 IEEE
DOI 10.1109/ICSE-SEET.2017.15

therefore we can conclude that all of them constitute learning outcomes of the course.

The rest of the paper is structured as follows. Section II discusses some related work. Our capstone course is described in Sec. III. The survey design, application and analysis are presented in Sec. IV. Section V discusses its results and the threats to validity. Finally, Sec. VI presents some conclusions and future work.

II. Related Work

Software engineering is an industry-oriented discipline; therefore, software engineering curricula need to prepare students to be ready to enter the world of industrial software development [3], however, young professionals are not always well prepared for this purpose. In order to address this issue, several universities have designed capstone courses, i.e., last year courses where all the knowledge acquired during the career has to be applied in practice.

One of the first reports of a software engineering capstone course was that of Moore and Potts [12]. They designed *The Real World Lab* in order to emulate an industrial organization. The course lasted for three quarters and students from the last two years were involved. Projects come from industrial sponsors who act as clients, providing consulting, reviewing, direction and resources. Students were suggested to use the *Mini-Task*, a development process based on the waterfall life cycle, however, after the first iteration they were allowed to change this process including methods and techniques they thought useful. Students answered a survey where they had to evaluate to what extent they thought they had a *real* experience and they feel better prepared for autonomously dealing with sizable projects, complexity, flexibility and uncertainty. Authors report that students feel much more confident and motivated after the course, reinforcing the value of a capstone course. Even though we have some insights about how students feel about our *Software Process* course, in this paper we are more concerned about measuring its learning outcomes.

Tvedt et al. [19] created the *Software Factory*, a two semester course based on the idea of a hands-on experience. There, students develop projects either from scratch or maintenance projects. Students enrolled in the *Software Factory* are assigned different roles according to their seniority. The goal of the course is to meet industry needs by educating computer science engineers that master technology and processes, ensuring a solid and lasting knowledge. They present a qualitative analysis intended to validate if they were reaching their goals. They found that all these goals were fulfilled. They also report that the beginning was hard on students and instructors, but they liked the experience and ended with a realistic perspective of the work of a software engineer. Our *Software Project* course is also hand-on, but it is less focused on teaching technology and processes but in letting students decide about these and other issues when facing a real project and exercising soft skills.

Germain et al. [8] proposed a capstone course called *Studio in Software Engineering* where all teams have to develop the same project using UPEDU [16], a customized version of RUP for education. Students enrolled in the course have already had a previous software engineering course. They reported that most teams effort was devoted to a few disciplines and that the learning curve of the process only let development begin after the middle of the course. The focus was on measuring how well students adopted the process, so they ended up with very good documentation but not as much software as expected. They concluded that a heavy process was not a good strategy for a capstone course. On this and other similar evidence we have included a series of agile practices and did not suggest a fixed process in our course.

Sebern [17] created the *Real World Lab* where students enroll after taking two other software engineering courses, so they already had some experience in developing software projects. He counted on real clients, but he proposes the use of reverse engineering of existing software instead of forward engineering because projects developed from scratch can either have a limited size or high risk of not achieving a functional product. Similar to our research approach, he conducted a survey about student perception of the learning outcomes; he asked about their perceived ability to: (1) document and plan changes, (2) work on a small team with specific roles, (3) apply a defined software process, (4) implement and test changes to a existing software, and (5) communicate project and process information to a real client. Students ranked 5 in a Likert scale their achieved ability for working in small teams, apply processes and implement and test changes; however, they ranked 4 their ability to document and plan changes and communication with the client. In summary, students were capable of linking together theory and practice, and there were benefits for both, students and clients that participated on the Lab. Even though the dimensions surveyed are similar to ours, the research strategy is different: he only evaluated perception after the course, and he used a Likert scale.

Mahnic [10] uses an agile approach on his capstone course, as we do. But, as he says that finding real clients is difficult, students work on teams developing a quasi-real project with user requirements provided by a domain expert playing the role of the Product Owner. Before the course students were already familiar with traditional software development methods. While students were enthusiastic about the agile approach, they sometimes missed a more detailed up-front design that would provide them the big picture so that development could be organized and planned more easily. We encourage the use of agile practices but, since we always have real clients, the clients themselves make sure that student teams work toward building the product they want without loosing their path.

Vanhanen et al. [20] reported a capstone course with industrial clients. They use and ad-hoc agile-based process and a mentor, who helps and gives advice to students during the whole semester. In this sense their course is quite similar to ours. Undergraduate students participate as developers while master students act as managers. Students considered the course stressful and laborious, but also extremely rewarding. Course feedback indicates that the course requires comparatively more effort per credit unit than most other courses. Similar to our

research approach, students answered a survey both, at the beginning and at the end of the course, about: familiarity with agility, knowledge about programming, project planning and management skills, effort estimation, software development process, teamwork, customer interaction, and communication skills. The dimensions that presented a larger improvement were familiarity with agility, knowledge about programming and communication skills. Even though these are not exactly the same dimensions we address, they can be mapped, however our study focuses on both, perceived value and perceived difficulty because these two characteristics of each dimension yield different learning outcomes.

Weissberger et al. [21] reported a capstone course where students have to work on software maintenance projects and the client is always the university. When they take this course, students are already familiar with software engineering concepts. They apply agile practices such as stand-up meetings, pair programing, burn-down charts, etc. They evaluate the course through a risk analysis of classmate schedules, concurrent work with other courses, college breaks, client schedule, learning a new framework and technical challenges. These dimensions are similar to those we consider. Their research approach is also similar: students rate the relevance of each of these issues and, a the end of the course, they analyze what actually happened. They found that teamwork related risks were those that affected the project results the most. Provided that we address value and difficulty of each dimension separately, we found that the value of teamwork did not grow significantly, but students found it really difficult.

Dunlap [5] conducted a study for measuring the observed increase of self-efficacy in students during a capstone course, by using a problem-based learning approach. He presents a qualitative analysis of open questions, where he asked students what they perceive they have learned during the course, what they think they still need to learn, and if they are confident to be able to deal with a real software project. The design, planning and continuous improvement of another capstone course on software engineering is presented by Stettina et al. [18]. They validate their findings with a survey, where they asks students about their satisfaction related to: the project, the teamwork, the communication within the project, the use of stand-up meetings and the use of meeting minutes. In both papers their study is similar to ours in several ways: they are based on a student survey, the sample size is almost the same, and they follow the same strategy of comparing pre and post course data. However, the former focuses on evaluating self-efficacy while the latter is interested in measuring the impact of inter-team stand-up meeting, on team satisfaction and coaching success. In our course we apply a series of agile practices: client on site, incremental development and deployment, and timeboxing, but other practices such as stand-up meetings, kanban boards, and planning poker are only suggested. In this work we also try to be more detailed in the sense of analyzing the perceived increase in consciousness that students achieve about the value and difficulty of correctly addressing different dimensions that may affect project success.

III. Capstone Course Setting

A. The Course

Software Project is a fifth year course for Computer Science Engineering students at the Universidad de Chile. It is the last mandatory course in their curriculum, and it is offered every semester. The curriculum includes a plethora of technical courses, including computer architecture, operating systems, computer networks, algorithms and data structures, computing theory, databases and programming languages. Also, the course is preceded by two courses about software engineering, one focused on methods and techniques, and the other on project development, mainly web-based information systems. But students have little or no experience on soft skills like project management, interpersonal communication promoting healthy and productive work environments, and understanding and satisfying actual business needs by negotiating with stakeholders.

We designed the course so as its main learning outcome is *that students internalize how relevant is having and developing critical soft skills to succeed in software development projects*. To achieve this outcome, we provide students a real-world environment to work in, but we support their progress by means of personalized tutoring and, when necessary, by intermediation with clients. The course is focused in four dimensions: technical challenge, negotiation with the client, project planning, and teamwork. While previous courses in the curriculum prepare the students to cope with the technical challenge, the other three dimensions are practically not covered or undeveloped. In this course we promote and encourage the use of agile practices to enforce exercising people skills, and to allow students succeed in building an actual usable product in a short period of time.

The learning objective of the course is *to provide students a controlled experience in real professional projects developed in-company for external clients*. The students work in teams of 4 to 7 people. Each team is assigned a project and a client to work for, to develop a software product in three iterations. Each team is tutored by an academic or Ph.D. student from the Computer Science Department. The tutor follows the progress of the project and provides guidance on technical, management and people issues. However, the tutor is not part of the team, does not take part in the product development and does not interact with the client. There is also an instructor, specifically the academic responsible for the course, that selects the projects, negotiates working conditions for students with the clients, coordinates and monitors all teams and tutors, and intervenes whenever major problems emerge.

The course has a duration of fifteen weeks and a workload of sixteen hours a week. Before the beginning of the course, the teams, tutors and projects are assigned and the iteration deadlines are defined. During the first week an industry expert gives a lecture on applying agile practices in real industry projects. Also, each team presents what they envision for the project to be developed, based on the project proposal and a first meeting with the client. During the remaining fourteen weeks, the teams work at the clients' office and participate

in a weekly meeting with the tutors and the instructor at the university. Every five weeks an iteration ends, and each team publicly presents the development experience and the developed product.

At the beginning and at the end of the course, each team answers a survey on the relative relevance of the four dimensions that we defined. The surveys and their results are discussed in Section IV. Also, at each presentation at the end of the iterations, students are encouraged to explain their retrospective analysis referring to these dimensions. Finally, at each weekly meeting, teams, their tutor and the instructor, discuss the progress of the project and how the four dimensions are being addressed and impacted by the practices adopted by each team.

B. The Teams

Students are randomly assigned to teams from 4 to 7 people, where the actual size depends on the total numbers of enrolled students in the semester. Team members have no defined roles, and they all share the responsibility for the success of the project. Teams are self-organizing as they assign tasks to members, and are self-managed as they hold each member accountable.

C. The Clients

Project proponents are private companies from any industry sector, governmental institutions and possibly university departments other than the Computer Science Department. Proposal submission is free and takes place a month before the semester starts. Each proposal describes the client organization and the project context, and states the problem to be addressed and an approximation to the solution.

The instructor is responsible for selecting which proposals to work with. The selection is based on the project complexity so as to exclude simple solutions and favor new clients whenever possible. The selected clients pay the Department a modest amount of money that is used for paying tutors and a per diem for students. The amount paid for the whole project is less than what a single graduate student would be paid in the industry, in average, for two months.

Each client provides the work environment to accommodate the assigned team, including office space and supplies, as well as development software, workstations and servers.

D. The Projects

The number of selected projects depends on the number of teams, and ranges from 2 to 6 each semester. Teams are randomly assigned to projects but supervised by the instructor to avoid any conflict of interest (e.g. a team member that is somehow related to the client). Tutors are randomly assigned to teams, also supervised by the instructor to avoid conflict of interest.

The project must follow a value-driven approach [1], focusing on what the client needs more than what the client asks for. Whenever needs and requests do not match, teams are encouraged to try to understand the actual client's motivation and to propose and explain alternative courses of action that the students consider to be better. The client is responsible for prioritizing the features to be included in the product under development, while the teams are responsible for estimating the effort required for the development. Both, the team and its client, negotiate the scope to be addressed in each iteration.

There are some mandatory agile practices for the teams to apply, such as timeboxing, client on site, and incremental development. Also, teams are encouraged to use some other agile practices such as daily stand-up meeting, Kanban boards and continuous integration. The industry expert talk at the beginning of the semester provides the students an insight on these practices, and tutors guide teams on applying them and possibly selecting others.

TABLE I
COURSE EVALUATION

Aspect	Iteration 1	Iteration 2	Iteration 3
Project management	Tutor	Tutor	Tutor
Product quality	Tutor	Tutor	Tutor
Product presentation	Instructor	Instructor	Instructor
Job performance	Students	Students	Not evaluated
Value of the solution	Not evaluated	Client	Client

E. The Evaluation

The course evaluation takes place at the end of each of the three iterations. The client, the instructor, the tutors and the students participate in the evaluation, each one assessing some aspects of the project: project management (tutor), product quality (tutor), product presentation and demo (instructor), job performance (students), and value of the solution (client) as described in Table I. Not every aspect is evaluated with the same weight in each iteration.

By evaluating these aspects, we quantify the perception of all stakeholders on how successful the projects are, measuring both the products and the development experience. Thus, these aspects are a measure on how successful are the practices being applied by each team, they allow students to adjust or change some of those practices at each iteration, and to analyze which kinds of skills need more focus in the remaining of the projects. By these means, we promote students to reflect on the need of critical soft skills and encourage them to develop whatever each team is missing.

IV. SURVEY

A. Survey Design

Students were asked to fill the data depicted in Table II by assigning a percentage to each of the four items: *technical challenge*, *negotiation with the client*, *project planning* and *teamwork*. The sum of the values for these four items must be 100 for both, value and for difficulty. The meaning of these values is the perceived relative value and difficulty of correctly addressing each dimension for the success of their project. At the end of the semester, the same student teams were asked to answer the same survey, but now considering

TABLE II
SURVEY INSTRUMENT

	Value	Difficulty
Technical challenge		
Negotiation with the client		
Project planning		
Teamwork		

their experience with the project. For this second survey, students do not have the information they provided for the one at the beginning of the semester in order to minimize bias. Additionally, we analyzed the comments that students anonymously provided in the university moodle-like platform at the end of the course looking for qualitative data that could reinforce or contradict the quantitative data. It is worth mention that providing comments is not mandatory, and therefore this information can be considered just anecdotal: there are around 6 to 10 comments each semester both positive and negative.

B. Data Collection

At the beginning of the last two semesters, each of the 7 teams that involved 38 students, was assigned a project and all of them were asked to answer the survey.

All projects were different in nature and complexity, but they were comparable in size. Project *Conicyt* involved the integration of several diverse legacy systems in a unique user interface that allowed transparent interaction. *SmartCities* required building a mobile application for providing guidance when using public transportation in Santiago, Chile. *Autofact* had to build an application that allow buyers to identify license plate numbers in car pictures. *NIC Labs* project developed a real-time visualizer for a DNS query log. The *Hitmap* project involved geo-referencing maps for real state. *Unholster* required building a recruiting and hiring system. Finally, *DAQ-EQ* referred to the development of a system integrating software and hardware for managing seismic sensors.

Tables III through VI show the initial and final perceived value and difficulty of correctly addressing technical challenge, negotiation with the client, project planning and teamwork for the success of the project. The last two columns show the variation in perception of value and difficulty of each dimension after the course. We also provide the mean and the standard deviation for each column.

C. Data Analysis

We ran a one-tailed dependent paired samples t-test in order to compare the students' perceptions before and after taking the course. There were 7 teams involved, and thus 6 degrees of freedom, and we used a value of $\alpha = 0.05$. Therefore $t_{6,0.95} = 1.943$. Tab. VII states the calculated t_C for the variation in the perceived value and difficulty of each dimension.

1) Technical challenge: A capstone course is intended to nail together the knowledge acquired during all previous courses. If addressing technical challenges correctly is initially perceived as a highly valuable dimension, it may mean that students are not able to foresee the relative relevance of other factors in the success of the project. Similarly, if after the project, this value drops, it would mean that they acknowledge that other issues might be more determinant; it would also show that students are already technically prepared and that they have the knowledge required for facing real world software projects.

H_{v1} *The perceived value of addressing technical challenges drops after the course.*

On the other hand, if dealing with technical challenges is perceived as difficult, it would mean that students do not feel prepared for professional work. But, if after the course, this perceived difficulty drops, it means that they realized they were actually well prepared for the challenge.

H_{d1} *The perceived difficulty of addressing the technical challenges drops after the course*

As can be seen in Tab. VII, the perceived value of addressing technical challenge drops substantially after the course and therefore we can confirm H_{v1}, but even though the perception of difficulty of addressing technical challenge drops in average, this cannot be considered statistically significant. We have found some comments in the university platform that reinforce these findings: "Our curriculum focuses mainly in theoretical knowledge; this course helps us put this knowledge into practice."

2) Negotiation with the client: Our curriculum includes two short professional internships where students work in a company mainly as programmers where they are assigned an activity they must develop with little room for designing and proposing their own solutions. So undergraduate students are not expected to be experienced in negotiation with real clients. However, we have found on the university platform some comments like: "This course provides very little for students with some working experience". Therefore, we wanted to find out how much students in general learned about negotiation with the client.

The mean variation of the perceived value of the negotiation with the client is -3.3, as can be seen in Tab. IV, so we state the following hypothesis:

H_{v2} *The perceived value of the negotiation with the client grows after the course.*

However, looking at Tab. VII, we realize that t_C is -1.3 for this dimension, and thus this hypothesis cannot be confirmed. On the other hand, the perceived difficulty seems to drop in Tab. IV.

H_{d2} *The perceived difficulty of the negotiation with the client drops after the course*

Table VII shows a t_C of 3.6, and therefore data is statistically significant to confirm this hypothesis: students are better prepared for this task than they thought before the course.

TABLE III
INITIAL AND FINAL PERCEPTION ABOUT VALUE AND DIFFICULTY OF CORRECTLY ADDRESSING TECHNICAL CHALLENGE

	Initial		Final		Variation	
Team	**Value**	**Difficulty**	**Value**	**Difficulty**	**Value**	**Difficulty**
Conicyt	35	20	24	14	11	6
SmartCities	28	25	10	27	18	-2
Autofact	40	50	30	50	10	0
NIC Labs	20	35	10	20	10	15
Hitmap	10	40	10	10	0	30
Unholster	23	15	20	25	3	-10
DAQ-EQ	50	43	25	35	25	8
				$\bar{X}_{TC}$	11.0	6.7
				S_{TC}	8.5	13.0

TABLE IV
INITIAL AND FINAL PERCEPTION ABOUT VALUE AND DIFFICULTY OF CORRECTLY ADDRESSING THE NEGOTIATION WITH THE CLIENT

	Initial		Final		Variation	
Team	**Value**	**Difficulty**	**Value**	**Difficulty**	**Value**	**Difficulty**
Conicyt	35	50	39	43	-4	7
SmartCities	25	25	37	10	-12	15
Autofact	20	20	20	15	0	5
NIC Labs	15	15	20	10	-5	5
Hitmap	40	40	30	20	10	20
Unholster	35	33	40	30	-5	3
DAQ-EQ	13	23	20	0	-7	23
				$\bar{X}_{NC}$	-3.3	11.1
				S_{NC}	6.9	8.1

TABLE V
INITIAL AND FINAL PERCEPTION ABOUT VALUE AND DIFFICULTY OF CORRECTLY ADDRESSING PROJECT PLANNING

	Initial		Final		Variation	
Team	**Value**	**Difficulty**	**Value**	**Difficulty**	**Value**	**Difficulty**
Conicyt	15	20	19	20	-4	0
SmartCities	28	40	30	40	-2	0
Autofact	20	20	25	20	-5	0
NIC Labs	35	35	45	30	-10	5
Hitmap	20	10	30	40	-10	-30
Unholster	25	27	20	30	5	-3
DAQ-EQ	14	20	30	20	-16	0
				$\bar{X}_{PP}$	-6.0	-4.0
				S_{PP}	6.8	11.7

3) Project planning: Students are taught how to plan a project in previous courses, and thus they are supposed to manage this skill when they enroll in our software project capstone course. But in this course it is the first time that clients put pressure on the students to finally get the software product they expect. According to the instructor's and tutors' perception, student teams feel this pressure and not always deal with it in the best way.

According to Tab. V, the mean perceived value for project planning is -6.0, so we state the following hypothesis:

H_{v3} *The perceived value of project planning grows after the course.*

Table VII states a value of t_C of -2.3, and thus we can reject the null hypothesis: students realize the actual value of project planning.

And about the perceived difficulty of addressing project planning, we hypothesize that it also grows.

H_{d3} *The perceived difficulty of addressing project planning grows after the course.*

Even though Tab. V shows a growth in the mean perceived difficulty, in Tab. VII we can see that t_C for this dimension is -0.9 and thus we cannot reject the null hypothesis. This difference may be due to the fact that students are taught project planning in previous courses and thus it did not resulted difficult; however, they just now realized how valuable it is for having the project under control and also probably manage the client's expectations.

4) Teamwork: Once every seven year, the career has to undergo a certification process. As part of it, surveys are conducted on students, former students, professors and employers. One of the major weaknesses found by employers

TABLE VI
INITIAL AND FINAL PERCEPTION ABOUT VALUE AND DIFFICULTY OF CORRECTLY ADDRESSING TEAMWORK

	Initial		Final		Variation	
Team	**Value**	**Difficulty**	**Value**	**Difficulty**	**Value**	**Difficulty**
Conicyt	15	10	18	23	-3	-13
SmartCities	19	10	23	23	-4	-13
Autofact	20	10	25	15	-5	-5
NIC Labs	30	15	25	40	5	-25
Hitmap	30	10	30	10	0	0
Unholster	18	25	20	15	-2	10
DAQ-EQ	23	15	25	45	-2	-30
				$\bar{X}_{TW}$	-1.6	-10.9
				S_{TW}	3.3	13.9

TABLE VII
T-TEST FOR DEPENDENT SAMPLES OF PERCEIVED VALUE AND DIFFICULTY

	Value				Difficulty			
Dimension	t_C	**p**	**Hypothesis**	**YES/NO**	t_C	**p**	**Hypothesis**	**YES/NO**
Technical challenge	3.4	0.01	H_{v1}	YES	1.4	0.09	H_{d1}	NO
Negotiation with the client	-1.3	0.14	H_{v2}	NO	3.6	0.01	H_{d2}	YES
Project planning	-2.3	0.04	H_{v3}	YES	-0.9	0.20	H_{d3}	NO
Teamwork	-1.3	0.86	H_{v4}	NO	-2.1	0.04	H_{d4}	YES

two years ago in their survey was that our professionals lack skills for teamwork. Therefore, we wonder if our course is teaching them something about this issue.

According to Tab. VI, there is an increase in the mean perceived value of teamwork, so our hypothesis is:

H$_{v4}$ *The perceived value of teamwork grows after the course.*

Apparently, as shown in Tab. VII, students have a realistic idea about the value of teamwork because there is not enough statistical evidence that indicate that its perceived value grows. Therefore we cannot reject the null hypothesis. However, the mean variation in perception about the difficulty of correctly addressing teamwork is much higher:

H$_{d4}$ *The perceived difficulty of addressing teamwork grows after the course.*

Table VII indicates a t_C of -2.1, and therefore we can reject the null hypothesis. Students tend to underestimate the difficulty involved in teamwork before the course, and therefore we can assume they will be more alert to this issue when facing a future project.

V. RESULTS

A. Discussion

We were able to validate some of our hypotheses. However, some of them, that we expected to be true, such as the perceived value of teamwork was not. Similarly, we did not expected that the perception of the value of project planning was not going to grow as it did.

Even though four of the hypotheses were corroborated, two of them had the most significant variation: the drop in the perception of the relative value of correctly addressing technical challenge, and the perceived difficulty of negotiating with the client. These two dimensions are the ones with the highest learning outcome. On the contrary, students had quite a clear idea about the difficulty of project planning, and thus the course made little difference.

B. Threats to Validity

According to the recommendation of Kitchenham et al. [9], we analyze three dimensions of threats to validity of our survey: content, criterion and construct validity.

1) Content validity: Content validity refers to how appropriate the survey instrument seems to be. We chose the surveyed dimensions on the effectiveness of software engineering capstone courses from other published reports, so we assume these dimensions are agreed to be relevant. Also, the perceived variation in knowledge is a commonly used measure of learning outcomes. So, we are following a standard approach supported by the research community in our evaluation.

2) Criterion validity: The only similar study we found in the literature is that of Vanhanen et al. [20]. They apply a similar survey but with a different strategy: they ask individual students about different learning dimensions with a Likert scale and then data is analyzed using the median instead of a t distribution. Nevertheless, results are consistent with our findings: dimensions that improve the most after the course where those related to technical issues and communication skills. We can consider this last dimension as a part of our negotiation with the client. These two dimensions were also those that improve the most in our survey. Thus, we can say that our instrument, is consistent with the findings of the only one we found, even with a much smaller sample.

3) Construct validity: Even though results tend to be fairly consistent for the sample considered, construct validity can only be assessed with a longer experimentation.

VI. Conclusions and Future Work

We designed our capstone course to provide students a concrete learning outcome: to internalize how relevant is having and developing critical soft skills to succeed in software development projects. Four years after agile practices were introduced in the course and its structure and operation became stable, we wonder whether our students were actually getting the declared outcome. That is, were they realizing the relevance of soft skills?

To answer this question, for the last two semesters we conducted a survey with seven participating project teams, involving 38 students, asking each team their perception on the relative value and difficulty on four dimensions: technical challenge, negotiation with client, project planning, and teamwork. Each team answered the survey twice: right after the project was assigned and the team had a first meeting with the client, and once right after the project ended. The survey allowed us to evaluate the variation between the initial and final perception of each team, and thus to capture what our students have learned from our course. Also, we applied a one-tailed dependent pair sample t-test to determine the statistical significance of the survey results.

We found out that the perceived relative value of soft skills grows while that of the technical challenge drops. We also found out that the perceived relative difficulty of soft skills grows in comparison to that of the technical challenge, except for the negotiation with the client whose perception of relative difficulty drops significantly. We cannot argue statistical significance for all of our findings. However, we have statistically significant evidence that, for the soft skills we have measured, the perceived relative relevance actually changes between the beginning and the end of the course, either due to a different relative value or to a different relative difficulty.

Specifically, we have evidence that the perceived relative value of the technical challenge drops by the end of the course. Thus, our students are already technically prepared and able to face real world software projects, and they acknowledge that soft skills are also determinant for the success of the projects. Also, our students are better prepared for negotiating with the client than as they thought before the course. However, we consider that having an instructor that hold accountable both teams and clients and that intermediates when necessary, facilitates and promotes successful negotiations. We also have evidence that students realize how valuable it is to have the project under control and also probably managing the client's expectations. Finally, students tend to underestimate the difficulty involved in teamwork before the course, and this course provides them evidence of the contrary. Therefore, we can assume that our students will be more alert to this issue when facing future projects.

Then, considering the research question of what can students get from a software engineering capstone course, we argue that, by means of our course setting, students can realize the relevance of soft skills, when compared with technical aspects, in order to succeed in software development projects. Moreover, we have statistically significant evidence to support this fact.

Nevertheless, our study leaves interesting open questions that require further research. Particularly, what aspects of the course structure are actually impacting the learning outcome? Is there a causality relationship between any aspect of practice of the course and the fact of achieving the learning outcome? Our assumption is that there are three aspects that are actually the most relevant. First, the fact that teams have a fixed amount of time to work in the project (timeboxing) makes clients to carefully select what to ask for. Second, the fact that each team works in the client's office, in direct contact of the client (client on site) makes students to be responsible in how they use their time and makes them accountable, both to the client and to team mates. Third, the fact that we enforce a non-hierarchical relationship between teams and clients, while providing an intermediary group of people that play the roles of external consultants (the tutors) and of arbitrator (the instructor) that not only intervenes when critical problems emerge, but also that demands and evaluates the fulfillment of the compromises of all involved parts, and that manage payments.

Acknowledgment

The work of Maíra Marques Samary was partially supported by the PhD Scholarship Program of Conicyt Chile (CONICYTPCHA/Doctorado Nacional/2012-21120544). This work was also partly supported by Project Fondef IDeA, grant: IT13I20010.

References

[1] Stefan Biffl, Aybüke Aurum, Barry Boehm, Hakan Erdogmus, and Paul Grünbacher. *Value-Based Software Engineering: Overview and Agenda*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[2] Chung-Yang Chen and P. Pete Chong. Software engineering education: A study on conducting collaborative senior project development. *Journal of systems and Software*, 84(3):479–491, 2011.

[3] Jianguo Chen, Huijuan Lu, Lixin An, and Yongxia Zhou. Exploring teaching methods in software engineering education. In *Computer Science & Education, 2009. ICCSE'09. 4th International Conference on*, pages 1733–1738. IEEE, 2009.

[4] Peter J Denning. Educating a new engineer. *Communications of the ACM*, 35(12):82–97, 1992.

[5] Joanna C. Dunlap. Problem-based learning and self-efficacy: How a capstone course prepares students for a profession. *Educational Technology Research and Development*, 53(1):65–83, 2005.

[6] Alan J. Dutson, Robert H. Todd, Spencer P. Magleby, and Carl D. Sorensen. A Review of Literature on Teaching Engineering Design Through Project-Oriented Capstone Courses. *Journal of Engineering Education*, 86(1):17–28, 1997.

[7] Gary Ford. The Progress of Undergraduate Software Engineering Education. *ACM SIGCSE Bulletin*, 26(4):51–55, December 1994.

[8] Éric Germain, Pierre N. Robillard, and Mihaela Dulipovici. Process activities in a project based course in software engineering. In *32nd Annual Conference on Frontiers in Education, FIE 2002*, volume 3, pages S3G–7. IEEE, 2002.

[9] Barbara Kitchenham and Shari Lawrence Pfleeger. Principles of Survey Research Part 4: Questionnaire Evaluation. *ACM SIGSOFT Software Engineering Notes*, 27(3):20–23, 2002.

[10] Viljan Mahnic. A capstone course on agile software development using Scrum. *IEEE Transactions on Education*, 55(1):99–106, 2012.

[11] Alejandra Martínez-Monés, Eduardo Gómez-Sánchez, Yannis A. Dimitriadis, Iván M. Jorrín-Abellán, Bartolomé Rubia-Avi, and Guillermo Vega-Gorgojo. Multiple Case Studies to Enhance Project-Based Learning in a Computer Architecture Course. *IEEE Transactions on Education*, 48(3):482–489, 2005.

[12] Melody M. Moore and Colin Potts. Learning by Doing: Goals & Experience of Two Software Engineering Project Courses. In *Software Engineering Education, 7th SEI CSEE Conference, Proceedings*, volume 750 of *Lecture Notes in Computer Science*, pages 151–164, San Antonio, Texas, USA, January 1994. Springer.

[13] Joint Task Force on Computing Curricula. *Software Engineering 2004: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering*. Computing Curricula Series. IEEE Computer Society - ACM, August 2006.

[14] Joint Task Force on Computing Curricula. *Software Engineering 2004: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering*. Computing Curricula Series. IEEE Computer Society - ACM, February 2015.

[15] Alex Radermacher, Gursimran Walia, and Dean Knudson. Investigating the skill gap between graduating students and industry expectations. In *Companion Proceedings of the 36th International Conference on Software Engineering*, pages 291–300. ACM, 2014.

[16] Pierre N. Robillard, Philippe Kruchten, and Patrick d'Astous. *Software Engineering Using the Upedu*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[17] Mark J Sebern. The software development laboratory: incorporating industrial practice in an academic environment. In *Software Engineering Education and Training, 2002.(CSEE&T 2002). Proceedings. 15th Conference on*, pages 118–127. IEEE, 2002.

[18] Christoph Johann Stettina, Zhao Zhou, Thomas Back, and Bernhard Katzy. Academic education of software engineering practices: towards planning and improving capstone courses based upon intensive coaching and team routines. In *26th International Conference on Software Engineering Education and Training (CSEE&T)*, pages 169–178, May 2013.

[19] John D Tvedt, Roseanne Tesoriero, and Kevin A Gary. The software factory: combining undergraduate computer science and software engineering education. In *Software Engineering, 2001. ICSE 2001. Proceedings of the 23rd International Conference on*, pages 633–642. IEEE, 2001.

[20] Jari Vanhanen, Timo OA Lehtinen, and Casper Lassenius. Teaching real-world software engineering through a capstone project course with industrial customers. In *Proceedings of the First International Workshop on Software Engineering Education Based on Real-World Experiences*, pages 29–32. IEEE Press, 2012.

[21] Ira Weissberger, Abrar Qureshi, Assad Chowhan, Ethan Collins, and Dakota Gallimore. Incorporating software maintenance in a senior capstone project. *International Journal of Cyber Society and Education*, 8(1):31–38, 2015.