Vanhanen, Jari; Lehtinen, Timo O.A.; Lassenius, Casper

Software engineering problems and their relationship to perceived learning and customer satisfaction on a software capstone project

Published in:
Journal of Systems and Software

DOI:
10.1016/j.jss.2017.11.021

Published: 01/03/2018

Document Version
Publisher's PDF, also known as Version of record

The Journal of Systems and Software 137 (2018) 50–66

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Software engineering problems and their relationship to perceived learning and customer satisfaction on a software capstone project

Jari Vanhanen[a,*], Timo O.A. Lehtinen[b], Casper Lassenius[a]

[a] Aalto University, Department of Computer Science, P.O. BOX 15400, FI-00076 AALTO, Finland
[b] Academy of Finland, P.O. BOX 131, FI-00531 Helsinki, Finland

ARTICLE INFO

Article history:
Received 20 June 2017
Accepted 5 November 2017
Available online 16 November 2017

Keywords:
Capstone project
Education
Learning
Customer satisfaction
Problems
Software engineering

ABSTRACT

In educational projects, having students encounter problems is desirable, if it increases learning. However, in capstone projects with industrial customers, negative effects problems can have on customer satisfaction must be considered. We conducted a survey in a capstone project course in order to study problems, learning and customer satisfaction related to eleven software engineering topics. On the average, students working in the managerial roles learned quite a lot about each topic, and the developers learned moderately, but the degree of learning varied a lot among the teams, and among the team members. The most extensively encountered problems were related to testing, task management, effort estimation and technology skills. The developers contributed quite a lot to solving problems with technology skills, but only moderately or less with other topics, whereas the managers contributed quite a lot with most of the topics. Contributing to solving problems increased learning moderately for most of the topics. The increases were highest with maintaining motivation and technology skills. Encountering problems with task management, customer expectations and customer communication affected customer satisfaction very negatively. When considering both learning and customer satisfaction, the best topics to encounter problems in were effort estimation, testing, and technology skills.

1. Introduction

In a capstone project, engineering students solve real-life problems in the context of a large, realistic project. As defined by Fincher et al. (2001) the capstone project aims "*to integrate and consolidate acquired concepts and skills through use on project work*". Capstone projects are commonly recommended and used in engineering education (ACM/IEEE CS, 2015; Pyster, 2009; Todd et al., 1995).

We have organized a capstone project course in software development at Aalto University for over two decades. On the course, teams of 7–9 students carry out real projects for real customers mostly from industry during a six-month time period. The learning process includes 1) conducting the necessary software engineering tasks from understanding the customer's problem to the delivery of functional software, 2) getting feedback on the intermediate and final results from the customer, and 3) reflecting on the used work practices with the team's mentor. The used software development process is iterative meaning that the working, feedback and reflection cycle is repeated several times. Real customers increase the realism of the projects and the student motivation. Unexpected challenges provide valuable learning opportunities that can be discussed among the teams.

In recent years, we have started to think about the relationship between learning and struggling with software engineering related problems in the projects. For example, does learning about requirements engineering challenges and practices suffer if the customer can express her needs very clearly, or if a team's mentor helps the team avoid all typical pitfalls in requirements engineering. Industrial projects aim at maximizing customer satisfaction, e.g., by avoiding problems as much as possible, whereas educational projects should focus on maximizing learning. Capstone projects with industrial customers need to focus on both learning and customer satisfaction. In order to find dozens of real topic proposals from industrial partners each year, the course must have a good reputation among them. However, learning may be in conflict with customer satisfaction if problems increase learning but decrease customer satisfaction. Thus, teachers of capstone project courses could benefit from a better understanding of what kind of problems students typically encounter in capstone projects, and

* Corresponding author.
E-mail addresses: jari.vanhanen@aalto.fi (J. Vanhanen), casper.lassenius@aalto.fi (C. Lassenius).

https://doi.org/10.1016/j.jss.2017.11.021
0164-1212/© 2017 The Authors. Published by Elsevier Inc. This is an open access article under the CC BY license. (http://creativecommons.org/licenses/by/4.0/)

how much the different problems affect students' learning and customer satisfaction.

In order to increase understanding of students' learning and customer satisfaction, and their relationship with software engineering problems in the context of a capstone project, we conducted a survey in our course. We focused on a select list of topics based on a previous qualitative study, in which we gathered hundreds of problems that our capstone project teams encountered during a previous run of the course (Vanhanen and Lehtinen, 2014). In this paper, we analyze how commonly the individual students working in different roles and teams encounter certain software engineering problems and how much they learn about the studied topics. Furthermore, we analyze the effect of the problems on learning and on customer satisfaction. We conclude the paper by analyzing the cost-benefit ratio of decreased customer satisfaction and increased learning for the various topics in order to identify for which topics struggling with the problems could be recommended.

Section 2 summarizes previous research related to problems, learning and customer satisfaction in capstone project courses, and to measuring learning. Section 3 introduces the software capstone project course at Aalto University. Section 4 presents the research method. Section 5 presents and discusses the results of our study, and the conclusions are drawn in Section 6.

2. Related work

Below we discuss previous research about software engineering problems, learning and customer satisfaction in capstone projects. Furthermore, as measuring learning is in central role in our study, we discuss the challenges related to it based on previous studies on the topic.

2.1. Software engineering problems, learning and customer satisfaction in capstone courses

The idea of learning through problems during a software development project course is not new. Dawson (2000) describes a project course, where twenty dirty tricks were used to disrupt the student's progress. The tricks were realistic challenges related to, e.g., requirements engineering, scheduling, work practices and tools. Based on informal feedback the course has been very educational, but the paper does not compare the learning outcomes to a setting where the desired problems are not encountered.

Previous empirical studies on encountered problems, students' learning outcomes, or their mutual relationship in the context of computer science capstone projects are rare, even though hundreds of papers on capstone projects have been written. Dugan (2011) reviewed a sample of about 200 papers on capstone projects in computing, but neither the learning outcomes nor the problems encountered by the project teams were among the common topics that emerged from the reviewed papers. Some of the papers discussed the learning theory behind the course, but even they did not evaluate the efficacy of the learning theory. Some customer satisfaction related aspects (meeting the requirements, taking the project into use, monetary savings from using the results) were discussed related to the used course evaluation techniques.

In our own literature searches, we found a few studies on capstone projects in computing that provide quantitative information on the problems encountered, or on students' learning (see Table 1), but we were not able to identify any papers with data about their relationship. Pournaghshband (1990) collected the five most serious problems encountered by each of their project teams, and lists the frequencies of the most common ones. All but one of the most common serious problems were related to teamwork and social issues, and the remaining one was related to testing. Ahtee and Poranen (2009) list the frequencies of realized risks, and Koolmanojwong and Boehm (2013) the frequencies of identified risks in their capstone projects. Risks can be considered as problems for a project at least if they realize. In these papers, the frequent risks included the lack of technology skills or lack of participation by the students, and covered also quality assurance, requirements engineering and scheduling.

Learning outcomes are seldom evaluated in detail in studies describing capstone courses in computing. Either the papers present no data, or the data is on a general level, typically mentioning that the students considered the course very useful or educational. Mahnic (2012) asked the students to evaluate their improvement in eight different skills such as programming, project planning and management, effort estimation, and team work on a scale of 1–5, where five meant maximal improvement. The median value of improvement was high (4) for practically all of the skills. Bruegge et al. (2015) report on students' improvement during the course in ten skills such as programming, design and version control, and on the changes in improvement over a four-year time period. During the latest run of the course, for each skill 70–84% of students "agreed" or "strongly agreed" that their skills improved. Broman et al. (2012) used an open question to survey the most important things for professional life learned during their capstone course. The most common answers mentioned by 6–10 of the 19 students were technical knowledge, time management, usefulness of agile methods, team communication, and collaboration.

Previous studies about customer satisfaction seem to be even rarer. Short remarks about the topic can be found in some capstone project case studies. For example, Goold (2003) reports 80% success rate over two years and 21 projects regarding schedule, scope and quality. Lack of project monitoring and control is reported as the main reason for the less successful projects. Clark (2005) reports about a course where the clients evaluate the developed software and the professionalism of the teams from several points of view. The results indicate high customer satisfaction, but Clark notes that the clients like to reward the students by giving full marks when in most cases they are not warranted.

2.2. Measuring learning

Measuring students' learning in a capstone course is complicated. The scope of topics a student can potentially learn is huge and can vary largely among the project teams, especially if each team has a different project topic and the freedom to choose their work practices, implementation technologies, and development tools. Furthermore, for many topics the learning goal is to achieve such a deep level of knowledge that a student is able to apply the knowledge in her future projects. Measuring the achievement of such knowledge can be more difficult than, e.g., measuring the knowledge of terminology.

Any standardized test of learning would have to contain very high-level questions due to the differences among the projects, and it would be difficult to come up with questions that would measure the deeper levels of knowledge. Furthermore, if a test is arranged only after the course or if the results of the project are used as a measure of learning, it is not possible to differentiate between what a student knew before the course and what she learned during the course. Exams involving writing essays would allow the assessment of deeper knowledge, but it would be laborious to cover even the central areas of software engineering, and difficult to evaluate the essays objectively.

A simple measure of learning that is commonly used in studies is to ask the students to evaluate their learning themselves. Unfortunately, the reliability of that method is limited. In a meta-synthesis of 22 meta-analyses the mean correlation between the self-evaluation of ability and performance outcome was found to

Table 1
Previous studies on encountered problems and learning.

Study	Sample	Focus
		Encountered problems by capstone teams
Pournaghshband (1990)	54 students, two courses	The most common serious problems listed by the students were: poor communication among the members (72% of students), poor leadership (61%), failure to compromise (56%), procrastination problems (54%), integration testing problems (44%), lack of cooperation (30%), lack of confidence (28%).
Ahtee and Poranen (2009)	76 projects, two instances of the course	The most commonly realized risks were related to: tools and skills (61% of projects), scheduling problems (61%), technology problems (53%), working and studying during the project (45%), motivation level low (36%), illnesses and social problems (34%), communication problems (32%), requirements (32%).
Koolmanojwong and Boehm (2013)	86 teams, five instances of the course	The most commonly identified risk categories were: 1. Architecture complexity, quality tradeoffs, 2. Personnel shortfalls, 3. Budget and schedule constraints, 4. COTS and other independently evolving systems, 5. Customer-developer-user team cohesion, 6. Requirements volatility, 7. User interface mismatch, 8. Process quality assurance, 9. Requirements mismatch, 10. Acquisition and contracting process mismatches.
		Learning outcomes
Mahnic (2012)	52 students in 13 teams	The median value of improvement in students' skills was high (4) on scale 1–5 for practically all of the skills that students evaluated: familiarity with agile approach, programming, project planning and management, effort estimation, "big picture" about software development process, team work, customer interaction, and communication.
Bruegge et al. (2015)	178 students in 40 projects, four instances of the course	After the latest instance of the course, 70–84% of students "agreed" or "strongly agreed" for each of the studied skills (requirements engineering, system design, modelling, programming, version control, release management, communication, team work, presentation, demo management) that their skills improved.
Broman et al. (2012)	19 students	The students listed technical knowledge, time management, usefulness of agile methods, team communication, and collaboration as the most commonly learned important things for professional life.

be 0.29 (Zell and Krizan, 2014). However, there is evidence that students' perceptions of learning might be more reliable when they assess deeper, more practical learning, which is similar to the scenario in capstone courses. In a study by Stehle et al. (2012), medical students' perceptions of learning on a course did not correlate ($r = -0.10$) with their results of a multiple-choice examination, but correlated strongly ($r = 0.50$) with the results of a test where they showed their practical skills with simulated patients or simulators.

3. The capstone project course

In the capstone project course at Aalto University, student teams develop real software for real customers. When this study was conducted, fifteen projects were carried out by teams of seven to nine students. Below, we describe the various stakeholders and roles involved, and the common software development process used in all the projects.

3.1. Project stakeholders and roles

Each project team has four to six Bachelor-level students working as developers. The course is scheduled for the third (last) year of their bachelor studies, and they have already studied many programming and computer science courses, and an introductory software engineering course.

Each project team has also three software engineering experts: a project manager, a quality manager, and an architect. They are typically Master-level software engineering students, who are participating the course for the second time. They have already taken the same capstone project course as developers, and thereafter studied some advanced software engineering courses. However, due to the limited number of the Master-level students some volunteers among the Bachelor-level students also work in these roles. The software engineering experts are responsible for project management, requirements engineering, quality assurance, and architectural design.

Each team may choose the three software engineering experts and three developers themselves. The course staff assigns the remaining developers evenly to all the teams based on their skills and interests. The course is a mandatory part of the studies for practically all the participants. Their years of presence at the university distributes quite evenly among three, four, five and more than five years.

The teacher looks for the customer candidates before the course begins. When the study was conducted, there were 26 topics available to be chosen by the fifteen teams. Most of the customers are from the industry, but there are also some real topics available from the university. During the projects, each customer actively participates in the requirements definition work as well as monitors the project progress. In the beginning of the project, the available effort is fixed to 125–200 h per student based on the credits each student aims at. Therefore, the customer must prune the scope of the project during the project according to the progress of the team.

The course staff consists of the course teacher and several mentors, who are previous students of the course. Each mentor typically guides two teams in issues related to the software development process. Before the projects begin, there are a few lectures related to the course arrangements, topic presentations, and used software development process. During the project the teacher arranges six experience exchange sessions, where individual students from all of the teams can join to discuss on problems with a particular theme. The project evaluation is based on both the work practices used and the results achieved. All projects have three phases, after which both the mentor and the customer gives their team points and concrete feedback.

3.2. Software development process

All the teams must apply the same predefined software development process framework. Some of the work practices have been defined strictly due to their educational or critical nature, but for many practices each team has a lot of freedom to customize them into their particular project. Many teams work a lot in a collocated manner having one to two weekly work sessions, but there are often some students who cannot attend all the sessions. A few teams work mainly in a distributed manner, e.g., due to the incompatible schedules of the team members.

All projects have three phases. Each phase ends in a project review with all the stakeholders including the customer, the mentor and the course teacher. In the project reviews, the project results are demonstrated, used work practices are discussed, and project status such as product quality and resource spending are presented. Before the project reviews, the teams must conduct a retrospective, where they analyze the used work practices.

The first phase lasts four weeks and focuses on setting up the project. It typically includes getting to know the team, deciding work practices, identifying the project goals, understanding why the system is built and for whom, identifying the most important requirements, identifying risks, drafting the architecture, choosing the implementation technologies, and setting up the development environment. During the first phase, the teams are required to write a project plan and a requirements document. However, documenting the individual requirements in detail is not required before they are chosen for the implementation in the upcoming phases.

The second and third phases last about six weeks and focus on developing and delivering features. These phases must be split into two or three sprints. Each sprint starts with sprint planning, where the sprint goals, deliverables and tasks are planned together with the customer. During each phase the teams aim at implementing, testing and delivering software that fulfils the chosen requirements to the customer.

Quality assurance is emphasized in the process framework. Each team must identify the most important quality goals, and choose and schedule the practices needed for achieving them. There is much freedom in choosing the practices, but each team must at least 1) prepare test cases for functional testing that cover at least half of the implemented system requirements, 2) perform a reasonable amount of unit level testing, 3) use a coding standard, 4) organize a code review for at least one critical component of the system, and 5) arrange peer testing with one other team at the end of the project using exploratory testing.

The process framework helps the teams define their development process more quickly than from scratch. It also tries to ensure that all teams get practical experience in using certain software engineering practices that are aligned with the educational goals of the course. Some of the required practices or documents can be unnecessary for fulfilling the customer's goals for the project, but from the teacher's point of view it is better to have some overhead in the process, e.g. a risk management process, if it provides valuable educational opportunities or decreases the rate of totally failed projects.

4. Research method

4.1. Background

This study builds upon our previous study about the problems encountered on a previous run of the same capstone project course (Vanhanen and Lehtinen, 2014). In that study, we conducted 2-hour-long retrospectives in eleven student teams twice during their projects. As a result, we identified hundreds of concrete software engineering problems encountered by the teams.

In this new study, we analyze problems, learning, and customer satisfaction in the same course. We focus on eleven software engineering topics (Table 2) with which problems were common according to our previous study, as well as our experience over the years. The selected topics cover most of the common software engineering topics, leaving out mainly the pure programming work where the problems are typically very project-specific due to the different technologies used in the projects.

Table 2
The studied software engineering topics.

Topic	Abbreviation
1. Gaining an understanding of the features desired by the customer	Features
2. Gaining an understanding of the quality requirements desired by the customer	Quality requirements
3. Managing the customer's expectations of the project scope	Customer's expectations
4. Planning and performing the testing	Testing
5. Getting all the developers reasonably skilled with the implementation technologies	Technology skills
6. Managing the versions of the source code	Version control
7. Estimating the effort of the project's tasks	Effort estimation
8. Managing which tasks to do next and how	Task management
9. Communicating within the team	Team communication
10. Communicating with the customer	Customer communication
11. Maintaining the motivation of the team members	Maintaining motivation

We used a questionnaire-based survey to collect the data. The topics were made comprehensible to the respondents by giving them simple, informative labels and by accompanying each of them with 2–5 representative, concrete problems (Table 3) selected from the data of the previous study.

4.2. Research goal and research questions

Our research goal is to better understand how much the students learn from a capstone project, and whether struggling with problems during the project affects their learning or customer satisfaction. As we are particularly interested in the role of problems, we focused on studying topics (Table 2) related to which students are likely to encounter problems.

We pose four research questions:

- RQ 1. How much do the students struggle with problems in their projects?
    - RQ 1a. To what extent do the students encounter problems in their projects?
    - RQ 1b. How much do the encountered problems affect the students' work?
    - RQ 1c. How much do the students contribute to solving the encountered problems?
- RQ 2. How much do the problems affect learning?
- RQ 3. How much do the students learn?
- RQ 4. How much do the problems affect customer satisfaction?

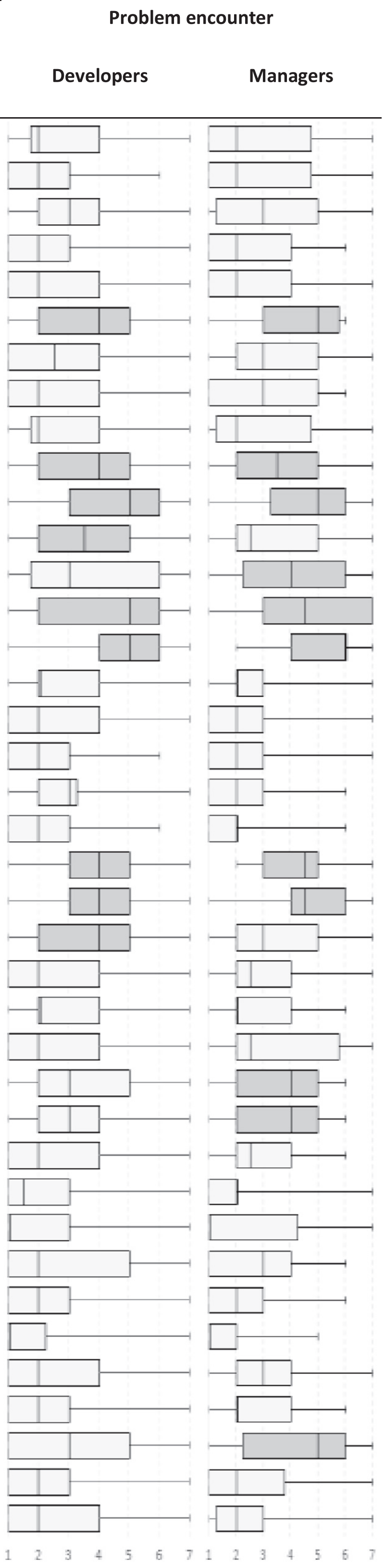
In RQ 1 and RQ 2 we study the individual students, but in RQ 3 we study the differences in learning both among the individual students and among the teams. In RQ 4, the analysis is done on the team-level only, as customer satisfaction is a team-level metric. For all research questions, we analyze how the student's role as a manager (project manager or quality manager) or a developer (architect or developer) affects the results.

In RQ 1a, we separately study each of the 39 concrete problems grouped under the eleven topics. In RQ 1b, RQ 1c, RQ 2 and RQ 3 the unit of analysis is a topic as a whole, and in RQ 4 we study both the concrete problems and the topics.

In RQ 1, we study struggling with problems from three points of view: encountering them, own work being affected by them, and contributing to solving them. In RQ 2 and RQ 3, we study learning about the purpose, challenges and work practices related to a topic. In RQ 2 and RQ 4, the effects of the problems are studied by calculating the correlation between struggling with the problems and learning or customer satisfaction.

Table 3
The extent of encountering problems. $N_{dev} = 88$, $N_{man} = 26$.

Topic	Problem
1. Features	The customer had poor understanding on the features that he or the users
	The customer had contradictory ideas
	The features were specified on too general a level
2. Quality requirements	First the customer wanted only features, but later started to require also quality
	The customer’s feedback of the realized software quality came late in the project
	Converting quality requirements into concrete tasks was difficult *
3. Customer’s expectations	The customer expected too low effort for the features
	Someone from the team promised too much to the customer
	The customer had difficulties in prioritizing the requirements
4. Testing	**Selecting the testing tools and practices was difficult**
	The developers took the testing tasks less seriously than coding tasks
	The requirements were specified on too general a level for supporting testing
	For a long time, the system was too unfinished for testing
	The amount of testing was lower than planned
5. Technology skills	**The team members were inexperienced with the implementation technologies**
	The team members' impl. tech. skills remained insufficient throughout the
	There was insufficient support within the team for learning the impl.
6. Version control	Changes to the source code were overwritten by other team members
	Merging conflicting changes in the source code was laborious
	The practices for using the version control tool were inadequate
7. Effort estimation	**Estimates for tasks that required learning activities were poor**
	Implementing tasks with desired quality required more effort than estimated
	Effort estimation was considered as an unhelpful activity
8. Task management	The backlogs were out-of-date
	The developers were unaware of their tasks
	The project management tool had inadequate support for the project
	The tasks were planned on too general a level
	Started tasks remained uncompleted
9. Team communication	There was lack of communication within the student team
	There was no comm. channel that would have been followed by all team
	The use of a foreign language caused difficulties for communication
	The meetings were inefficient
10. Customer communication	The customer was slow in responding to team's questions
	The team delayed communicating negative issues to the customer
11. Maintaining motivation	The team members ignored the agreed work practices
	The team members avoided taking responsibility
	Paid work overrode the course project *
	Team building activities were insufficient
	The team members lacked motivation



Scale: “not at all” (1) – “very much” (7)
Statistical significance of the difference between the roles based on Mann–Whitney U Test: *p < .05, **p < .01

4.3. Data collection

The study was conducted as a mandatory questionnaire-based online survey with closed questions in the end of the projects. The survey contained several questions, which were repeated on a separate page for each of the eleven topics. All the questions had the same 7-point scale where only the extreme ends were labeled as “not at all” and “very much”. The respondent had to choose one of the seven points within the extreme ends, and there was no “I don’t know” option available.

The questions asked regarding each topic were:

1 To what extent did you personally encounter the following problems related to [topic X]?
   - 2–5 concrete problems and a generic *“Other problems with [topic X]”* line followed this question

2 Please, consider ALL the problems that your team encountered with [topic X]:
   - How much did these problems affect your own work?
   - How much did you personally contribute to solving these problems?

3 Please, consider the purpose, challenges and work practices regarding [topic X]:
   - How much did you learn about this topic?
   - How much expertise did you have about this topic before the project?

Furthermore, we collected demographic data, including the student’s team number, the role in the team, and the amount of previous IT work experience as years. The time spent on answering each page was automatically recorded.

Of the 127 students on the course, 124 answered the questions, giving a response rate of 97.6%. However, we excluded the data of ten respondents because they clearly had used too little time for reading the questions to be able to answer them honestly, resulting in a final response rate of 89.8%.

Customer satisfaction was evaluated by the customer on a 15-point scale, where 13–15 points meant “*exceeds expectations*”, 10–12 points meant “*meets expectations*”, 7–9 points meant “*slightly below expectations*” and so on. The evaluation was conducted in the end of the project in a discussion with the teacher, who tried to unify the expectation levels and evaluation criteria among the different customers.

4.4. Data analysis

We coded the responses of the main questions between 1 (“not at all”) and 7 (“very much”). We analyzed the data quantitatively using the IBM SPSS Statistics 23 software (IBM Corp., 2015). As the data was on ordinal scale, and in many cases the distributions were skewed, we analyzed medians instead of means. The medians characterize the average student on the course better than the means. Due to the ordinal scale, we also used non-parametric methods. The relationships between the variables were analyzed using Spearman’s rank correlation (r_s). The statistical significance of the various differences between the manager and developer roles were analyzed using the Mann–Whitney U-test.

4.5. Limitations

We analyze the threats to the validity of our study using the four aspects of the validity discussed by Runeson and Höst (2009): construct validity, internal validity, external validity, and reliability.

4.5.1. Construct validity

The limitations with the self-evaluation of learning were discussed in Section 2.2. In our case, learning is mainly related to the achievement of practical skills, which is a context where self-evaluation can be more reliable than in general cases as found by Stehle et al. (2012).

Answering the questionnaire was a mandatory part of the course, but the students were allowed to fill it online and unsupervised. We expected that not all the students would take it seriously, and therefore logged the time taken to fill the survey in order to filter out responses that were most likely to be given without reading the questions.

4.5.2. External validity

Our study focused on a pre-selected set of software engineering topics. There are certainly also other relevant topics about which the students learned, or with which the problems affected customer satisfaction. Based on our study, it is not possible to say anything about topics that we did not study.

The specific arrangements and contexts of capstone courses are likely to affect how much the students encounter problems and learn, and how much the various problems affect customer satisfaction. Generalizing the results related to these variables should be limited to courses that are similar enough to our course. We believe that the most impactful similarities are related to the development process used in the projects, and having real customers for the projects. On the other hand, there is no clear reason why the effect of struggling with problems on learning would be specific to our course, and thus those results could be generalizable to many other capstone courses.

4.5.3. Internal validity

In our study, we examine the causal relationships of struggling with problems to learning and customer satisfaction. While correlation does not imply causation, it is not far-fetched to assume that struggling with problems can facilitate learning or that encountering problems in a project can cause decreased customer satisfaction. However, we acknowledge that there can be linking variables, such as the amount of effort spent working with a certain activity, which can affect both the amount of struggling with problems and learning.

4.5.4. Reliability

In our study, the data analysis is not dependent on the researcher as we had only quantitative data that was analyzed using statistical methods. However, there are concerns related to the questionnaire.

In our questionnaire, all the main concepts (topic, problem, and learning) were somewhat abstract, and we asked the respondents to evaluate rather vague aspects related to them (the amount of struggling with the problems in various ways, the amount of learning). When using a questionnaire, there is a risk that the respondents do not understand the questions in the same way as the researcher expects or that they do not possess all the information necessary to answer the questions (Foddy, 1993). We tried to mitigate these risks by piloting and iteratively improving the questionnaire. We also tried to communicate the topics clearly by providing clear labels and concrete sample problems for each topic.

Furthermore, in our questionnaire, the higher end of the response scale used in all the main questions was not tied to anything concrete, and the respondents could have interpreted “very much” in different ways. With regard to learning we considered also an alternative formulation (“much more than on any other course”). However, it may have been too modest a level for the capstone course, and it would have also depended on which courses each student had taken previously.

5. Results and discussion

5.1. Problems encountered

In RQ 1a we studied the extent to which the students encountered problems in their project. In the survey, each student evaluated the extent she encountered each of the 39 problems listed on the survey. The box plots of the responses are shown in Table 3 grouped by the eleven topics. The most common problems are bolded. The scale is "not at all" (1) – "very much" (7). All the "*Other problems with [Topic X]*" items had low values (Mdn ≤ 3.0), and have been excluded from the table.

The managers encountered 16 problems to a larger extent than the developers whereas the developers encountered only five problems to a larger extent than the managers, when considering the median values. This difference between the roles may reflect the managers' broader sense of responsibility of and participation to the project especially when considering activities that are not directly related to programming. However, the distributions of the responses for each problem are large, and the differences between the roles are statistically significant only for two problems: 1) Converting quality requirements into concrete tasks was difficult, and 2) Paid work overrode the course project.

The majority of the problems were not common (Mdn ≤ 3.0) even though they were selected to the study because we knew based on previous data (Vanhanen and Lehtinen, 2014) that they had taken place at least in some teams. However, each problem was encountered extensively by at least some individuals. The different project contexts and different areas of responsibilities within the teams could explain the differences in encountering problems among individuals.

Thirteen of the 39 problems were encountered more commonly (Mdn ≥ 3.5 for the developers and/or managers). These problems are spread among six of the eleven studied topics (see the bolded lines in Table 3). Next, we list the most common problems grouped by the topic, and discuss potential reasons for their commonness. The reasons mentioned are speculative in the sense that we do not have qualitative data to back them up. However, they are based on our long-time experience in running the course.

Technology skills (topic 5) involves the most common problem for both roles: the team members were inexperienced with the implementation technologies. Its commonness is not surprising, because the projects often require the students to use technologies (programming languages, web frameworks etc.) that have not been taught in their previous courses, and only some of the students have used them in their work or hobby projects. Furthermore, in order to balance the competences among the teams, the team forming process aims at assigning the least experienced students evenly into all of the teams.

Testing (topic 4) involves five of the thirteen most common problems. Problems in the attitude to testing and the amount of testing as well as in selecting testing tools and practices were commonly encountered in both roles. The low amount of testing is a likely consequence of the negative attitude to testing complemented with the other problems that complicate performing testing. The system being too unfinished for testing was a common problem for the managers but less so for the developers. It can be that the developers understand better that some forms of testing, such as unit testing, can be done before the system or a feature is completely finished. Too general requirements for supporting testing was a common problem for the developers only. It can be that the managers understood the requirements better as they had more contacts with the customer. On the other hand, it may be that the managers did not understand how concrete the requirements need to be to support testing. Converting quality requirements into concrete tasks under *quality requirements* (topic 2) is also closely related to testing, and was a common problem for both roles.

Effort estimation (topic 7) involves three of the most common problems. These include poor estimates for tasks that have strict quality requirements or that involve learning activities. Furthermore, estimation was considered an unhelpful activity. It may be that in a fixed budget and fixed schedule project (or iteration) where mainly critical features are implemented in the priority order, the delivered software will be the same regardless of any attempts to estimate the task efforts. Furthermore, in the course projects it is quite common that a new team uses new technologies to develop a system to an unfamiliar domain using a new development process making estimates very inaccurate. Therefore, the developers may not see any benefits in trying to estimate the tasks.

Task management (topic 8) involves two of the most common problems. Started tasks remained uncompleted, and tasks were planned on too general a level. The factors that complicate task effort estimation can also complicate planning concrete tasks, and if tasks are not concrete, they are also more difficult to complete. Tasks remaining open may also indicate the lack of regular effort invested in the project. We have often seen that some students or even some teams increase their weekly effort to the project only closer to the end of the phases or the end of the project, if ever.

Maintaining motivation (topic 11) involves one of the most common problems: paid work overrode the course project. It was very common for the managers, but clearly less common for the developers (Mdn = 5.0 vs. Mdn = 3.0, p = .024). The reason for the large difference can be that the managers were generally older students, who had slightly more IT work experience in the end of the course (Mdn: 2.0 vs. 1.0 years, averages: 2.4 vs. 2.0 years, *t*-test two-tailed: p = .479).

None of the most common problems is related to the remaining topics: *features, customer's expectations, version control, team communication,* and *customer communication*. It is surprising that the problems related to the customer are rare. The most common customer related problem is that the features were specified on too general a level (Mdn = 3.0 for both roles). We expected that problems related to *customer's expectations* and *customer communication* would be more common, but it may be that our actions in recent years have somewhat decreased them. We have been able to gather much more project topic proposals than there are teams, which has meant that only the most committed customers have found a team.

As a summary, only a few problems were encountered extensively by a majority of the students. The most extensively encountered problems are mainly related to *testing, task management* and *effort estimation*. Furthermore, inexperience with the implementation technologies under *technology skills* and paid work overriding the course project under *maintaining motivation* were also among the most extensively encountered problems. Similar problems were reported also in previous studies (Pournaghshband, 1990; Ahtee and Poranen, 2009; Koolmanojwong and Boehm, 2013) but any detailed comparison among the studies is impossible due to the different problem classifications used in the studies.

5.2. Effect of the problems on own work and contribution to solving them

Next, we look at the effect the problems had on the students' own work, and the students' contribution to solving the problems. In the survey, each student evaluated both of these aspects for each topic considering all the problems encountered related to the topic as a whole. The medians of the responses are shown in Table 4 and the distributions in Fig. 1. The problems affected the managers' work slightly more than that of the developers for al-

Table 4
Previous expertise, the effect of the problems on own work, problem solving contribution, learning, and correlation between problem solving contribution and learning. $N_{dev} = 88$, $N_{man} = 26$.

Topic	Previous expertise		Effect of the problems on own work		Problem solving contribution		Learning		Correlation between problem solving contribution and learning			
	Dev.	Man.	Dev.	Man.	Dev.	Man.	Dev.	Man.	Dev.		Man.	
	Median	Median	Median	Median	Median	Median	Median	Median	r_s	p	r_s	p
1. Features	3.0*	4.0*	3.0*	4.0*	4.0*	**5.5***	5.0*	5.0*	.36	.001	-0.04	.860
2. Quality requirements	3.0	4.0	2.0**	4.0**	4.0*	5.0*	4.0**	5.0**	.46	.000	.08	.715
3. Customer's expectations	3.0*	4.0*	2.0**	4.0**	4.0**	5.0**	4.0**	5.0**	.36	.001	.46	.017
4. Testing	3.0	3.0	3.0**	**4.5****	3.0**	5.0**	4.0*	5.0*	.45	.000	.33	.097
5. Technology skills	3.0	3.0	**4.0**	3.0	**5.0****	3.0**	5.0*	4.0*	**.55**	**.000**	.38	.059
6. Version control	**4.0**	**4.5**	2.0	2.0	3.0	3.0	4.0	4.0	.29	.007	-0.02	.921
7. Effort estimation	3.0	3.0	3.0**	4.0**	3.0**	4.0**	4.0	5.0	.43	.000	.46	.019
8. Task management	3.0	4.0	3.0	3.0	4.0**	5.0**	4.0	5.0	.24	.026	.43	.029
9. Team communication	**4.0**	4.0	2.0*	3.5*	4.0*	5.0*	4.0	5.0	.17	.116	.40	.043
10. Customer communication	3.0*	4.0*	2.0**	3.0**	2.0**	5.0**	4.0**	5.0**	.45	.000	.45	.022
11. Maintaining motivation	3.0	4.0	2.0*	3.0*	3.0**	4.5**	4.0**	5.0**	.31	.003	**.58**	**.002**
All topics	*3.0***	*4.0***	*2.0***	*3.0***	*4.0***	*5.0***	*4.0***	*5.0***	*.39*	*.000*	*.35*	*.000*

Scale: "not at all" (1) – "very much" (7)
Statistical significance of the difference between the roles based on Mann–Whitney U Test: $^{*}p < .05$, $^{**}p < .01$

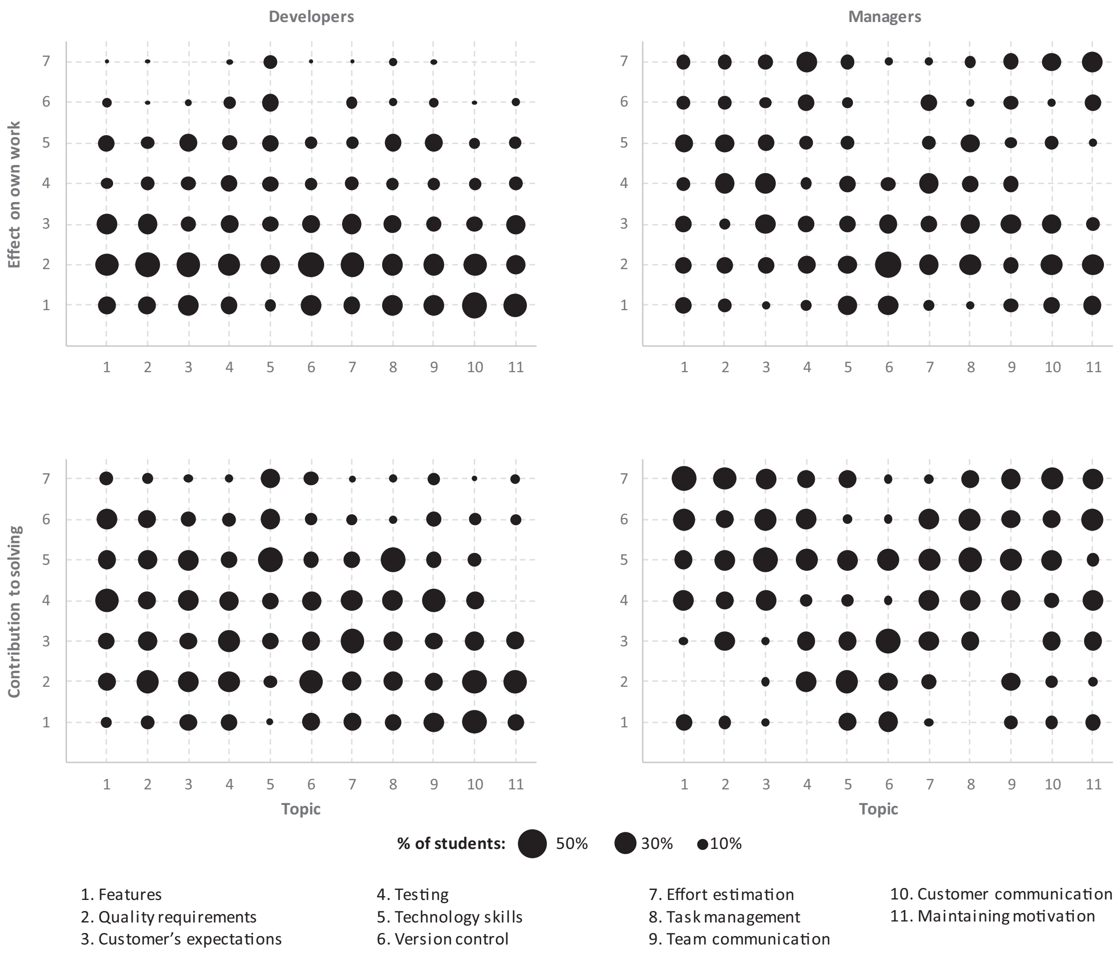


Fig. 1. The distributions of the effect of the problems on own work and contribution to solving them. $N_{dev} = 88$, $N_{man} = 26$. Scale: "not at all" (1) – "very much" (7).

most all topics. However, the effects were rather small for both roles, with the medians being 3.0 vs. 2.0. The managers also contributed more than the developers to solving the problems (median 5.0 vs. 4.0). The differences between the roles are statistically significant for almost all topics with both aspects (Table 4). The differences between the roles support our conception that the managers have a more wide-ranging sense of responsibility of and participation in the project, in particular related to non-technical topics.

5.2.1. Effect of the problems on students' own work

The developers' work was affected the most by problems related to *technology skills* (Mdn = 4.0), but for the other topics their work was not affected much ($2.0 \leq \text{Mdn} \leq 3.0$). The distributions (see Fig. 1) show that *technology skills* clearly differs from the other topics by having the highest number of developers whose work was affected a lot by the problems, and the lowest number of developers whose work was not affected at all. The distribution is also the most even one among all topics and ranges over the whole scale. It means that the differences among the developers were largest with this topic. As mentioned in Section 5.1, inexperience with the implementation technologies and various testing related problems were the most common problems encountered by the developers. If a developer is not capable of performing programming tasks adequately, it naturally has a major effect on the work. Fortunately, the skills generally did not remain insufficient throughout the projects (see Table 2).

The managers' work was affected most by problems related to *testing* (Mdn = 4.5), but all other topics followed closely ($3.0 \leq$ Mdn ≤ 4.0) except *version control* (Mdn = 2.0). As mentioned in Section 5.1, testing involved the highest number of the most commonly encountered problems. The strong effect of the testing related problems is probably related to both the large number of the encountered problems and the fact that poor quality really gets the project manager and quality manager embarrassed when demonstrating the software to the customer at the end of a sprint.

It is interesting to note that even though both the developers and managers encountered problems related to testing to a large extent (see Section 5.1), the developers considered being affected by them clearly less than the managers. Combined with the finding that the developers clearly took the testing tasks less seriously than the coding tasks (see Table 2), it seems that the developers did not care about software quality as much as the managers.

5.2.2. Contribution to solving the problems

The effect of the encountered problems to a student's own work and her contribution to solving them correlate moderately. When considering all the topics as a whole, the Spearman correlation for the developers is 0.44 ($p < .001$) and for the managers 0.56 ($p < .001$). These correlations indicate that the students contribute more to solving problems that affect their own work than other problems.

The developers contributed most (Mdn = 5.0) to solving the problems with *technology skills*. It is the only topic where the developers' contribution was higher than that of the managers. The developers contributed least (Mdn = 2.0) to solving problems related to *customer communication*, which is typically a topic that the managers are mainly responsible of in these projects.

The managers contributed quite a lot to solving problems encountered with most of the topics, medians being 4.5–5.5. Their contribution was low (Mdn = 3.0) only for two topics: *technology skills*, and *version control*. The reason for the low contribution can be that the problems encountered with these two topics did not affect much the managers' work, and they are the most technical topics among the studied topics.

Both the managers and developers encountered estimation related problems to a high degree, and their work was affected by them, but neither the managers nor developers contributed much solving these problems. It may be that the students did not expect that such problems would affect the project results, and did not consider estimation an important activity. The activity was still done and problems were encountered, because the course's process framework enforced the students to estimate task efforts due to educational purposes.

5.3. Learning

In RQ 2, we studied the effect of problems on learning. In RQ 3, we studied how much the students learned related to each topic. In the survey, we defined learning as covering the purpose, challenges and work practices related to a topic. We asked the students about their perceived learning related to each topic on the scale "not at all" (1) – "very much" (7).

5.3.1. Effect of the problems on learning

There is a moderate, statistically significant Spearman correlation ($0.29 \leq r_s \leq 0.58$) between learning and problem solving contribution for most of the topics and both roles as shown in Table 4. These correlations are stronger than the ones between learning and being affected by the problems ($r_s = 0.39$ vs. $r_s = 0.13$ for the developers, and $r_s = 0.35$ vs. $r_s = 0.26$ for the managers, when considering all the topics as a whole). They are also stronger than between learning and encountering problems, both when considering individual problems and aggregated problems of a topic. Thus, it seems that the students learned more if they contributed to solving problems and were not only affected by them. However, as the correlations between learning and problem solving contribution are only moderate or smaller for each topic, some other factors than struggling with problems also affected learning. For example, just working on a topic, even without encountering explicit problems, could have caused learning. Unfortunately, we did not collect data on this factor, and cannot analyze its relationship with learning.

Problem solving contribution increased learning most with *maintaining motivation* ($r_s = 0.58$, $p = .002$ for managers) and *technology skills* ($r_s = 0.55$, $p < .001$ for developers). On the other hand, for three topics the correlation is close to zero for the managers. These topics are *features, quality requirements*, and *version control*. For the developers, the weakest correlation is 0.17 ($p = .116$) related to *team communication*.

Based on the quantitative data it is impossible to say why contributing to solving problems increased learning more for any particular topic than for others. *Features* and *quality requirements* may be topics, which require lots of attention from the managers in every project, and therefore the managers may learn a lot of them whether or not they encounter explicit problems related to these topics. On the other hand, *motivation* and *technology skills* may not get much attention in an inherently highly motivated and technically competent team, and learning about these topics may be higher in projects where they involve explicit problems.

5.3.2. Amount of learning

Generally, the managers learned quite a lot about each topic median per topic being typically 5.0, and the developers learned moderately median being typically 4.0. Similar high levels of learning in capstone courses were reported in other studies (Mahnic, 2012; Bruegge et al., 2015). The developers learned more only about *technology skills*, and both roles learned the same amount about *features* and *version control*. Many of the differences in learning between the roles are statistically significant (Table 4).

It is interesting that the managers learned more than the developers about the studied topics, even though they had already done the capstone project once as developers. It must be noted that the studied topics are biased toward areas where the managers were more involved than the developers. The developers worked mainly with programming related activities that were not covered much in this study, and they may have learned a lot about them.

The amount of learning varied only minimally among the topics for both roles. For the managers, the only topics with a slightly lower amount of learning (median of 4.0 instead of 5.0) are the most technical topics: *technology skills*, and *version control*. This is in line with the managers' lower problem solving contribution with these topics. For the developers, the only topics with a slightly higher amount of learning (median of 5.0 instead of

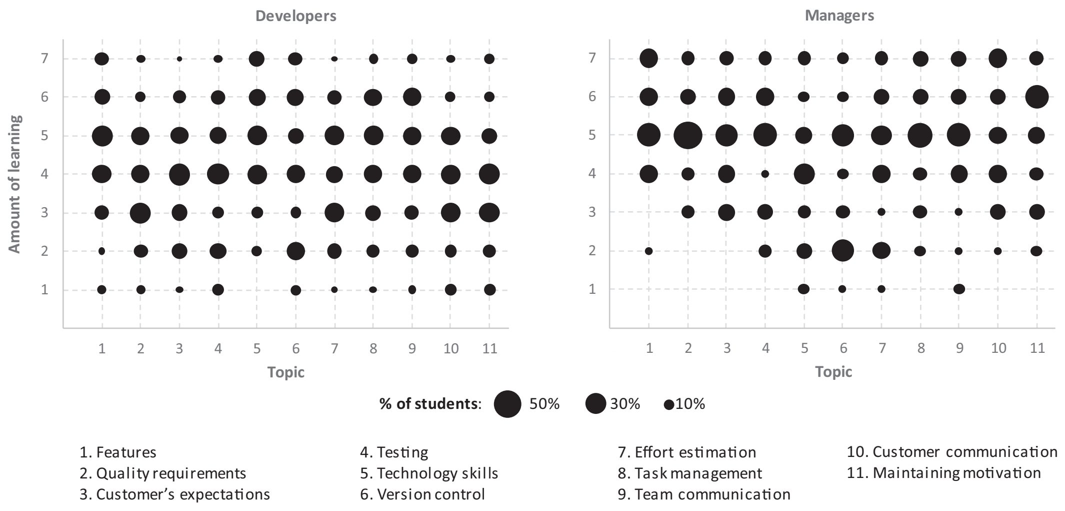


Fig. 2. The distributions of learning. $N_{dev} = 88$, $N_{man} = 26$. Scale: "not at all" (1) – "very much" (7).

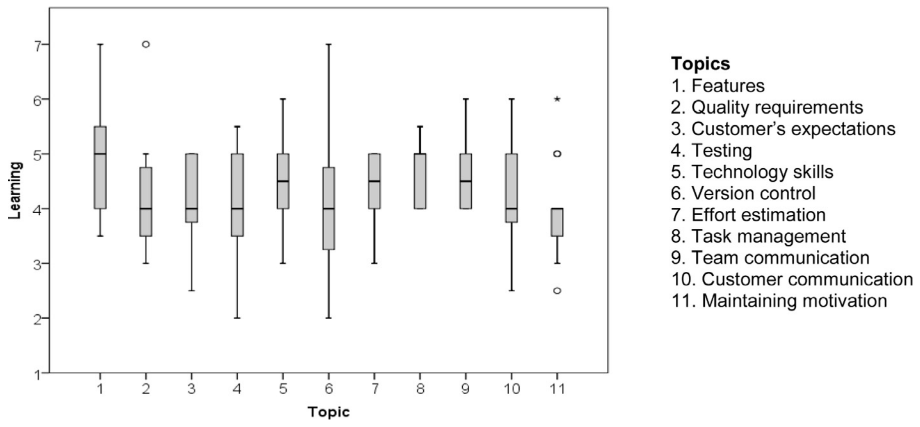


Fig. 3. The distributions of the teams' learning. $N_{teams} = 15$. Scale: "not at all" (1) – "very much" (7).

4.0) are *features* and *technology skills*. Minimal variation in learning among the topics is surprising as the effect of the problems on learning varied among the topics. This means that for some reason other factors affected learning less when the effect of the problems on learning was higher.

5.3.3. Differences in learning among the students and among the teams

The equal median values for learning about most of the topics does not mean that an individual student learned the same amount of all topics. Most of the students (84%) actually learned a lot (≥ 6) at least about some topic. For each topic, the amount of learning varied a lot among the students as can be seen in Fig. 2. For both roles and most of the topics, the interquartile range is 2.0–3.0 units and the responses distribute to the whole scale from "not at all" to "very much". The only topics with a smaller distribution are the managers' learning of *features* and *quality requirements*. For these topics, the majority of the managers learned quite a lot (≥ 5), and the proportion of responses under 4 was particularly low compared to the other topics. It can be that understanding what is to be done in a project is something that almost every project manager and quality manager must be deeply involved in when software is made for an external customer. The other customer related topics (*customer's expectations, customer communication*) have some lower responses for learning, which may be because a couple of customers participated less actively to the project than the other customers.

The amount of learning varied a lot also among the teams. A team's amount of learning is defined here as the median of the team members' amount of learning, and thus emphasizes more the learning of the developers (about six per team) than that of the managers (two per team). For many topics there was a team or teams where the majority of the students learned a lot ($6.0 \leq Mdn \leq 7.0$), but also totally opposite teams ($2.0 \leq Mdn \leq 3.0$) as can be seen in Fig. 3.

Each team has a unique profile w.r.t. learning of the topics. The profiles of three example teams are shown in Fig. 4 to characterize the differences. We can see that Team 2 learned about most of the topics less than the other two teams. We can also see that the topics about which each team learned most, are different for each team. For example, Team 4 learned very much about *features* and *version control*. On the other hand, Team 14 learned a lot about *technology skills*, but only little about *testing* and *customer communication*.

The large differences in learning among the teams are likely to be due to the unique nature of each project (e.g. customer, domain, technologies, team members), which means that different activities were emphasized in each project. For example, in some projects it may have been difficult to understand what features the system should have whereas in other projects the technical

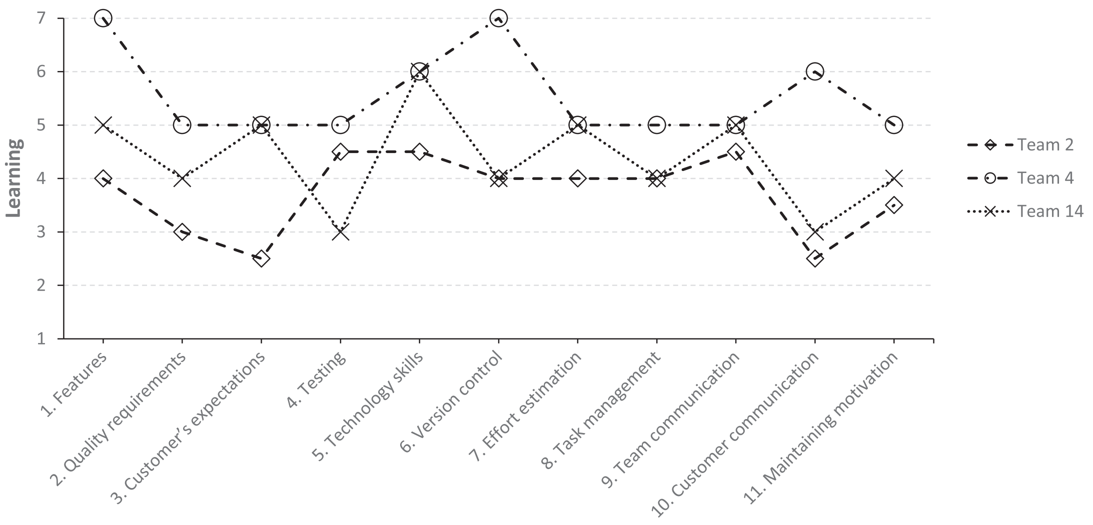


Fig. 4. The learning profiles of three example teams. Scale: "not at all" (1) – "very much" (7).

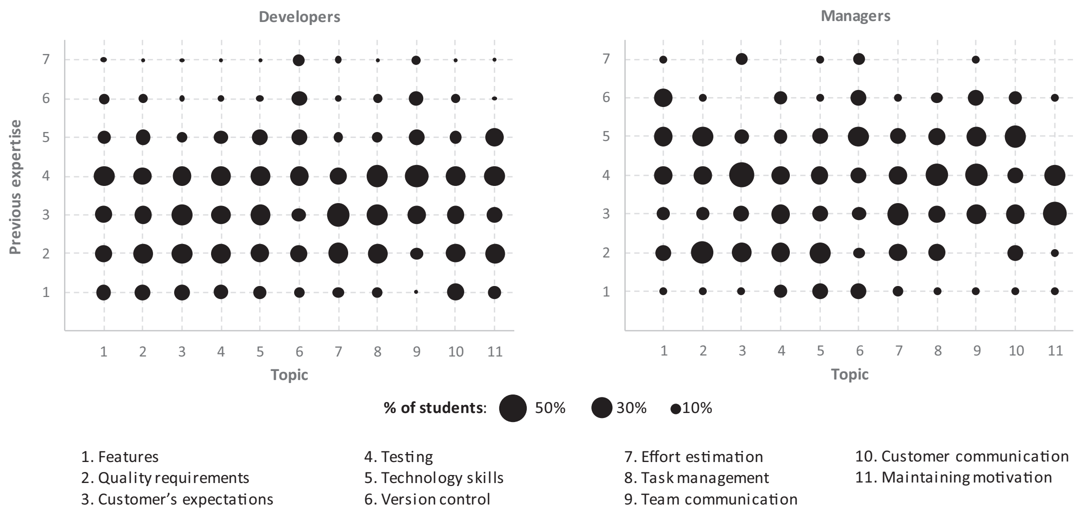


Fig. 5. The distributions of the students' previous expertise. $N_{dev} = 88$, $N_{man} = 26$. Scale: "not at all" (1) – "very much" (7).

implementation work may have been the most difficult part. Furthermore, with the customer related topics or with testing, only a couple of students may have taken the main responsibility, and the median of the team members' learning was low.

The amount of learning varied also even among the developers of the same team. The detailed responsibilities and interests of each student within a team may cause these differences. For example, backend and front-end developers may have had different amounts of customer communication needs during the project, which may have affected learning of that topic.

5.3.4. Previous expertise and learning

We expected that the amount of previous expertise with a topic may affect the amount of learning, see Fig. 5. However, for most of the topics for both roles the correlations between the amount of learning and previous expertise of a topic are small ($r_s < |0.3|$) and not statistically significant. Thus, in general the differences in previous expertise do not explain the differences in learning. It can be that it is always possible to learn more about software engineering topics, or that the generally low expertise of the students (Mdn = 3.0) did not yet limit the potential for learning more.

However, there are three exceptions where the correlation between learning and previous expertise is moderate and statistically significant ($p < .01$): *version control* for both roles ($r_s = -0.37$ for developers, $r_s = -0.50$ for managers), and *testing* ($r_s = 0.30$ for developers). All the students, but especially the managers, learned either quite a lot or only a little about *version control* (Fig. 2). It is a topic where the use of a tool is in a central role. A possible explanation for the negative correlations could be that those who were already familiar with the used Git tool perhaps did not learn much and vice versa. *Testing* ends up quite often being the responsibility of only one or two students at least what comes to setting up advanced testing tools, or performing other levels and types of testing besides unit testing. A possible explanation to the positive correlation between learning and previous expertise about testing could be that students who already knew more about testing volunteered to take the responsibility of testing, and thereby learned more about it than the other team members.

5.4. Effect of the problems on customer satisfaction

In RQ 4, we studied the relationship between the problems and customer satisfaction. Each customer gave a satisfaction score on a 15-point scale at the end of the project. The distribution of the given points was very even — between 12 and 15 points — except for a team, whose client was rather disappointed, which received only 9 points. Below, we analyze the relationship between encountering problems and customer satisfaction, both at the level of individual problems and topics.

5.4.1. Correlations between the problems and customer satisfaction

We analyzed the relationship between the problems and customer satisfaction at the team level, since customer satisfaction was a team-level metric. We defined the extent to which a team encountered a problem as the median value among the team members' encounter with the problem.

There are twenty problems for which the correlation between a team's encounter with the problem and customer satisfaction is statistically significant ($p < .10$) at least for the managers or developers (bolded in Table 5). All of these correlations are at least moderate ($|r_s| > 0.40$), and almost all are negative meaning that encountering problems decreases customer satisfaction. The only exception is the problem that effort estimation was considered as an unhelpful activity. It has a positive correlation ($r_s = 0.54$, $p = .039$) with customer satisfaction. In this context, estimation may really be an unhelpful activity, and it may be that the more enlightened students understand this, whereas the other students only follow course instructions without questioning their benefits for the project.

5.4.2. Correlations between the problems related to each topic and customer satisfaction

As learning was studied per topic, we analyze also the effect of the problems on customer satisfaction per topic. Above we defined the extent a team encountered a problem as the median of the team members' encounter with the problem. Here we use the median among a team's encounter with each of the problems related to the topic. The extent that the teams encountered problems with each topic and their correlation with customer satisfaction are shown in Table 6. For many topics, the correlations are weak meaning that the problems with those topics did not have much effect on customer satisfaction. However, for three topics, the correlations are stronger than |0.5| and statistically significant ($p < .05$) for either or both roles (bolded in Table 6). These topics are *task management, customer communication*, and *customer's expectations*. Below we discuss these three topics in more detail.

Problems related to *task management* decreased customer satisfaction if the developers ($r_s = -0.84$, $p < .001$) or the managers ($r_s = -0.66$, $p < .008$) encountered them. All the concrete problems related to *task management* have a strong negative correlation with customer satisfaction (Table 5). The strongest correlations are related to inadequate project management tool and tasks remaining uncompleted. For both roles these two problems have the strongest correlation ($|r_s| > 0.70$) among all the problems of all the topics. Tasks remaining uncompleted may mean that a team defines too large or unclear tasks that are difficult to close. However, it may also be an indicator of a more critical problem where a lazy team uses only a proportion of the allocated effort to progress the tasks, which would be a more likely explanation for the strong negative effect on customer satisfaction. The strong effect of poor project management tool is surprising and we do not have any explanation for it. The teams chose the tools they used themselves and there were many different tools used.

Problems related to *customer communication* decreased customer satisfaction if the developers encountered them ($r_s = -0.68$, $p = .005$). All the concrete problems related to *customer communication* have a strong negative correlation with customer satisfaction, if the developers encountered them (Table 5). The correlation is strongest for delaying communicating negative issues to the customer ($r_s = -0.64$, $p = .011$). Such a behavior may decrease possibilities for making corrective actions or for lowering the customer's expectation level during the project, and thus decrease customer satisfaction in the end of the project.

Problems related to *customer's expectations* decreased customer satisfaction if the developers ($r_s = -0.47$, $p = .075$) or the managers ($r_s = -0.52$, $p = .046$) encountered them. Someone from the team promising too much for the customer and Other problems with the topic both have a moderate correlation with customer satisfaction for both roles (Table 5). Furthermore, customer satisfaction suffered if the developers encountered the problem that the customer expected too low effort for the features, but not if the managers encountered it. It may be that the managers did not understand so well the required effort either, and did not identify this problem so reliably. All these problems show that customer satisfaction is not only about what you deliver but how that relates to the customer's expectations. A team can either increase or decrease these expectations during the project.

5.5. Cost-benefit ratio of the problems

5.5.1. Overview

From the teacher's point of view decreased customer satisfaction is the main cost of problems in capstone projects whereas increased learning is the only benefit of problems. We define the costs of problems as the correlation between the extent a team encountered problems with a topic and customer satisfaction. The benefits are defined as the correlation between a student's contribution to solving problems related to a topic and the student's learning of the topic. The cost-benefit ratio of struggling with the problems is shown in Fig. 6 separately for each topic and role. Topics close to the upper-right corner have a strong positive effect for learning but only a weak negative effect for customer satisfaction whereas the lower-left corner represents the opposite situation. Below, we list the best and most harmful topics for struggling with problems, and discuss how common they are in our course.

5.5.2. Good problems

The best topics to struggle with problems are *effort estimation, testing*, and *technology skills* which are educational for both roles, and do not decrease customer satisfaction more than a little at most. Furthermore, problems related to *quality requirements* are educational to the developers and do not decrease customer satisfaction when either of the roles encounters them. Problems related to *customer communication* are educational to both roles, but only the managers should struggle with them, or otherwise customer satisfaction decreases a lot. Below we analyze how much the students struggled with these good problems, and give some ideas on how the course arrangements may support encountering these problems.

Contribution to solving *effort estimation* related problems was moderate for both the developers (Mdn = 3.0) and the managers (Mdn = 4.0). In our course, we require that each team estimates the effort of all planned tasks in the beginning of each sprint. Generally the students comment that they don't see much benefits from estimation, and the situation is made worse as the estimates are generally poor especially in the early phases of the projects. The fact that the estimation problems do not affect customer satisfaction at all supports also its perceived uselessness. However, it is an important skill to learn for real projects, and struggling with estimation problems seems to increase learning about it, which justifies enforcing it on the course. If estimation was not enforced, many teams might not do it at all.

The developers contributed somewhat (Mdn = 3.0) to solving problems related to *testing* whereas the managers contributed rather lot (Mdn = 5.0). Furthermore, for the managers the distribution is bi-polar because in some teams only either the project manager or the quality manager contributed a lot and the other

Table 5
The correlations between each team's encounter with each problem and customer satisfaction. $N_{teams} = 15$.

Topic	Problem	Correlation between problem encounter and customer satisfaction			
		Developers		Managers	
		r_s	p	r_s	p
1. Features	The customer had poor understanding on the features that he or the users needed	−0.33	.228	−0.44	.105
	The customer had contradictory ideas	−0.27	.323	−0.23	.410
	The features were specified on too general a level	−0.36	.192	−0.09	.739
	Other problems	−0.17	.548	−0.23	.417
2. Quality requirements	First the customer wanted only features, but later started to require also quality	−0.26	.353	.07	.813
	The customer's feedback of the realized software quality came late in the project	−0.19	.486	−0.09	.751
	Converting quality requirements into concrete tasks was difficult	−0.05	.854	−0.02	.953
	Other problems	−0.26	.355	−0.04	.893
3. Customer's expectations	The customer expected too low effort for the features	−0.17	.548	**−0.48**	**.068**
	Someone from the team promised too much to the customer	**−0.54**	**.038**	−0.40	.142
	The customer had difficulties in prioritizing the requirements	−0.26	.349	−0.16	.558
	Other problems	**−0.46**	**.086**	**−0.59**	**.021**
4. Testing	Selecting the testing tools and practices was difficult	−0.02	.958	.10	.736
	The developers took the testing tasks less seriously than coding tasks	**−0.54**	**.039**	−0.34	.209
	The requirements were specified on too general a level for supporting testing	−0.33	.234	.16	.562
	For a long time, the system was too unfinished for testing	−0.20	.483	−0.24	.385
	The amount of testing was lower than planned	−0.21	.444	**−0.47**	**.080**
	Other problems	−0.37	.179	−0.20	.482
5. Technology skills	The team members were inexperienced with the implementation technologies	.17	.554	.14	.628
	The team members' impl. technology skills remained insufficient throughout the project	−0.20	.484	−0.31	.263
	There was insufficient support within the team for learning the impl. technologies	**−0.47**	**.074**	−0.03	.922
	Other problems	−0.14	.606	−0.27	.334
6. Version control	Changes to the source code were overwritten by other team members	−0.28	.315	.20	.483
	Merging conflicting changes in the source code was laborious	.25	.371	.15	.594
	The practices for using the version control tool were inadequate	**−0.47**	**.079**	−0.08	.769
	Other problems	**−0.45**	**.096**	−0.14	.610
7. Effort estimation	Estimates for tasks that required learning activities were poor	−0.28	.305	−0.13	.651
	Implementing tasks with desired quality level required more effort than estimated	−0.43	.107	−0.38	.160
	Effort estimation was considered as an unhelpful activity	**.54**	**.039**	**.46**	**.084**
	Other problems	−0.06	.821	−0.28	.306
8. Task management	The backlogs were out-of-date	**−0.55**	**.033**	−0.41	.124
	The developers were unaware of their tasks	**−0.56**	**.031**	−0.38	.157
	The project management tool had inadequate support for the project	**−0.84**	**.000**	**−0.71**	**.003**
	The tasks were planned on too general a level	**−0.56**	**.032**	.03	.909
	Started tasks remained uncompleted	**−0.72**	**.002**	**−0.77**	**.001**
	Other problems	**−0.57**	**.028**	−0.31	.260
9. Team communication	There was lack of communication within the student team	−0.37	.181	−0.19	.494
	There was no comm. channel that would have been followed by all team members	**−0.47**	**.079**	−0.27	.326
	The use of a foreign language caused difficulties for communication	−0.22	.438	−0.04	.899
	The meetings were inefficient	−0.27	.327	−0.35	.206
	Other problems	−0.33	.237	−0.25	.369
10. Customer communication	The customer was slow in responding to team's questions	**−0.49**	**.066**	**−0.63**	**.012**
	The team delayed communicating negative issues to the customer	**−0.64**	**.011**	−0.09	.753
	Other problems	**−0.59**	**.020**	−0.31	.257
11. Maintaining motivation	The team members ignored the agreed work practices	−0.41	.127	−0.43	.108
	The team members avoided taking responsibility	−0.28	.316	−0.36	.188
	Paid work overrode the course project	−0.18	.519	−0.04	.897
	Team building activities were insufficient	**−0.58**	**.022**	**−0.49**	**.066**
	The team members lacked motivation	−0.32	.246	**−0.48**	**.070**
	Other problems	−0.38	.161	−0.36	.185

one only little. In our course, we require that each team plans and executes various testing activities on different testing levels (see Section 3.2). We believe that the low contribution by the developers is due to their generally low participation to testing related activities especially what comes to planning the testing and performing other than unit testing. Such activities may end up being the responsibility of either or both of the managers and only some developer.

The developers contributed to solving problems related to *technology skills* rather lot (Mdn = 5.0) and it was more than they did for any other problems (Table 5). Our course design where most of the developers must use technologies that are unfamiliar

Table 6
The teams' encounter with problems related to a topic, and correlation between problem encounter and customer satisfaction. $N_{teams} = 15$.

Topic	Problem encounter at the team level		Correlation between problem encounter and customer satisfaction			
	Developers	Managers	Developers		Managers	
	Median	Median	r_s	p	r_s	p
1. Gaining an understanding of the features desired by the customer	2.25	2.25	−0.34	.213	−0.27	.334
2. Gaining an understanding of the QRs desired by the customer	2.25	2.50	−0.22	.441	.06	.832
3. Managing the customer's expectations of the project scope	2.00	3.25	−0.47	.075	**−0.52**	**.046**
4. Planning and performing the testing	3.50	4.00	−0.27	.323	−0.20	.485
5. Getting all the developers reasonably skilled with impl. technologies	2.50	2.50	−0.29	.293	−0.25	.378
6. Managing the versions of the source code	2.25	2.00	−0.44	.100	.06	.829
7. Estimating the effort of the project's tasks	3.50	4.00	−0.14	.614	−0.06	.822
8. Managing which tasks to do next and how	2.75	3.25	**−0.84**	**.000**	**−0.66**	**.008**
9. Communicating within the team	2.00	2.50	−0.32	.246	−0.39	.151
10. Communicating with the customer	2.00	2.00	**−0.68**	**.005**	−0.24	.388
11. Maintaining the motivation of the team members	2.00*	3.00*	−0.38	.164	−0.46	.084

Scale: "not at all" (1) – "very much" (7)
Statistical significance of the difference between the roles based on Mann–Whitney U Test: $^{*}p < .05$, $^{**}p < .01$

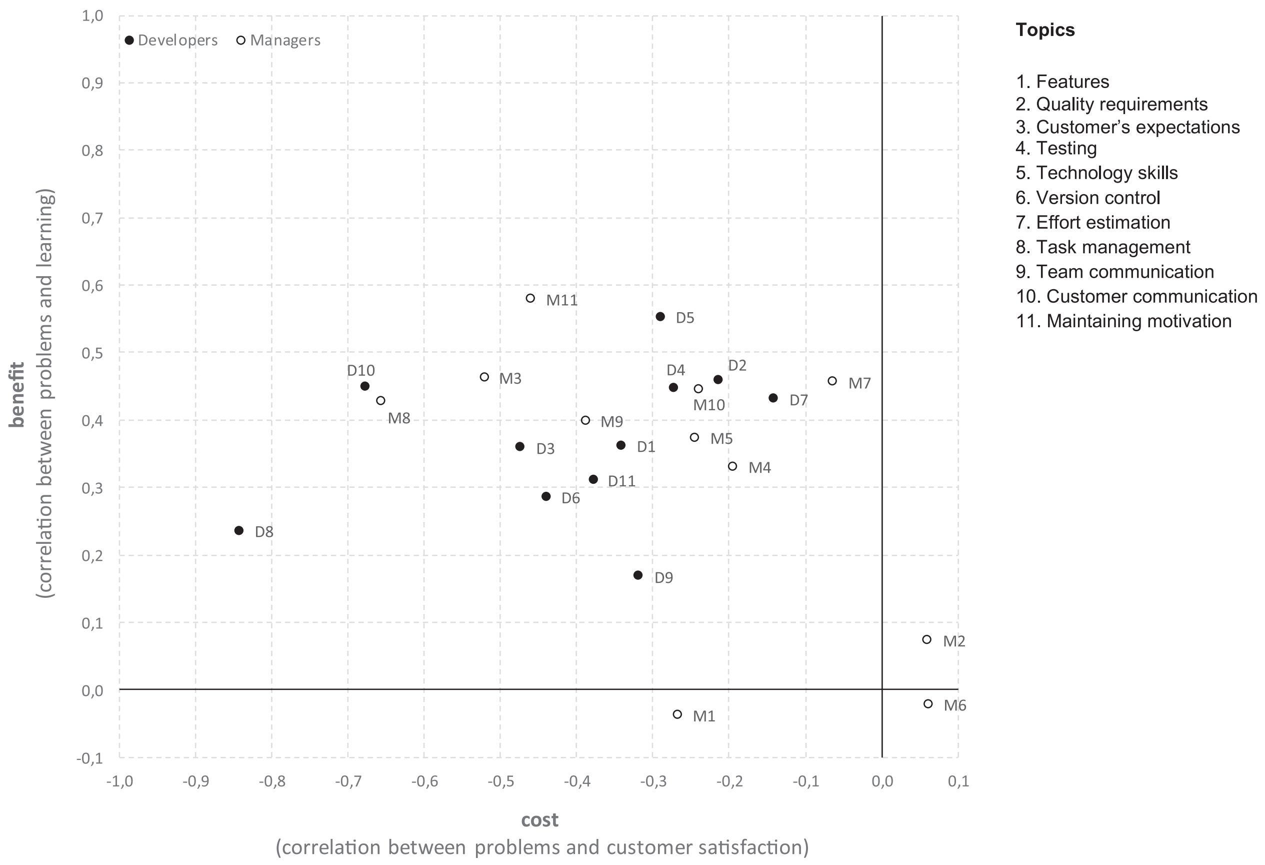


Fig. 6. The cost-benefit ratio of struggling with problems related to the different topics.

to them forces encountering these problems. On the other hand, the managers' contribution to solving these problems was lower (Mdn = 3.0) and it was the lowest of all the topics for them. For the managers the distribution is bi-polar (Fig. 1) meaning that some did contribute a lot but many contributed very little. Based on our experience, we know that only some of the managers are technically competent or interested in the technology aspects in the projects, and the differences in contribution may reflect this.

The developers contribute moderately (Mdn = 4.0) to solving problems related to *quality requirements*. In our course, each team is required to explicitly identify the main quality attributes of the developed system early in the project. Thus, the teams pay at least some attention to them during the project, and see challenges related to prioritizing them with the client and to ensuring their achievement.

The managers contributed a lot (Mdn = 5.0) to solving problems related to *customer communication*. In our course, real customers with real problems are closely engaged in the projects through the iterative development process. More detailed specification of the features is done an increment at the time ensuring continuous communication need with the customer.

5.5.3. Harmful problems

The worst situation is when the developers encounter problems with *task management* (topic 8). Such problems decrease customer satisfaction very much and don't have much educational value. In our course, the developers do not encounter them much, medians being $\leq$ 3.0 for all the problems (Table 3). When such problems take place, e.g., the developers don't know clearly what they should do next, the developers may end up doing less important tasks, or nothing at all.

Encountering problems with *customer expectations* (topic 3) is debatable because they are educational, but decrease customer satisfaction quite a lot for both roles. The same applies for the managers with *task management* (topic 8) and *maintaining motivation* (topic 11), and for the developers with *customer communication* (topic 10).

5.6. Practical implications

Our results help the teachers of capstone project courses make informed decisions on whether they should try to increase or decrease problems with a particular topic in their course. Preventing all problems is impossible, but through well-designed course arrangements and support to the students, something can be done. Increasing problems artificially is easier to achieve.

We recommend that the decision on artificially increasing or actively decreasing problems should be made per topic and it should be based on 1) the main learning goals of the course, and 2) the cost-benefit ratio of struggling with the problems related to the topic. If a topic is among the main learning goals of the course, it may be justified to ensure that the students struggle with problems even if they had somewhat negative effect to customer satisfaction as long as they are educational. On the other hand, if a topic is not among the main learning goals and/or struggling with problems has low educational value, the teacher should make all she can do to prevent the problems.

However, one should be careful when using problems as a learning mechanism. With most of the topics, solving problems correlated only moderately with learning meaning that much of learning is explained by other factors, such as just working around a topic, and learning will take place even if no problems were encountered. We have seen how a single, difficult problem in some area, such as motivation, may endanger the whole project. In such a situation, a team must invest lots of effort and attention to a single problematic topic, which means learning related to other topics suffers. Problems that involve such a risk should be avoided in all cases. Furthermore, even when the teacher tries to prevent some problems, some teams will usually still encounter them. Sharing the experiences of these unfortunate teams to the other teams can increase and harmonize learning across the teams without sacrificing customer satisfaction in all the teams. We have successfully used experience exchange workshops on particular topics for increasing learning.

Our results regarding learning show also that an idea of having common learning goals for all students is unrealistic in a setting with heterogeneous projects, and individual roles and responsibility areas. Rather, each student will have a unique learning experience whose match with her own learning goals depends on how careful she is when choosing a project, and her own role and responsibilities in the team. This emphasizes the importance of having each student pay attention to the desired learning goals in the beginning of the course, and consider them when choosing the team, role and responsibility areas in the project.

6. Conclusions and future work

This study aimed at increasing understanding of how much the students learn and struggle with software engineering problems in their capstone projects, and whether struggling with problems affects their learning or customer satisfaction. These viewpoints are important because capstone projects with industrial customers need to focus on both learning and customer satisfaction. However, if problems increase learning but decrease customer satisfaction, they are conflicting goals.

The study was conducted as an online survey to all the 127 students in our capstone project course, and the response rate was 89.8%. The questionnaire covered eleven software engineering topics, and 39 concrete problems categorized under the topics. All the questions related to learning and struggling with the problems used the same response scale: "not at all" (1) – "very much" (7).

In RQ 1, we studied how much the students struggled with problems in their projects. The majority of the studied problems were not encountered commonly. For only 13 of the 39 problems the median value for the encounter was $\geq$ 3.5 for the managers and/or developers. These problems were mainly related to the following topics: *planning and performing testing, managing which tasks to do next and how,* and *estimating the effort of the project's tasks*. Furthermore, inexperience with the implementation technologies and paid work overriding the course project were also among the most common problems. The managers struggled with the studied problems slightly more than the developers. It can be that the managers have a more wide-ranging sense of responsibility of and participation in the project, in particular related to non-technical topics.

The effect of the problems on the students own work were rather low for both the managers (Mdn = 3.0) and developers (Mdn = 2.0) when considering all the topics as a whole. The developers' work was most affected by problems related to *getting the developers skilled with the implementation technologies* (Mdn = 4.0), but with the other topics their work was not affected much by the problems ($2.0 \leq \text{Mdn} \leq 3.0$). The managers' work was most affected by problems related to *planning and performing the testing* (Mdn = 4.5), but all the other topics followed close ($3.0 \leq \text{Mdn} \leq 4.0$) except *managing the versions of the source code* (Mdn = 2.0).

Contribution to solving problems was moderate both for the managers (Mdn = 5.0) and developers (Mdn = 4.0) when considering all the topics as a whole. The developers contributed most (Mdn = 5.0) to solving the problems with *getting the developers skilled with the implementation technologies* and least (Mdn = 2.0) to *communicating with the customer.* The managers contributed quite a lot to solving problems encountered with most of the topics medians being 4.5–5.5. Their contribution was rather low (Mdn = 3.0) only for two topics: *getting the developers skilled with the implementation technologies,* and *managing the versions of the source code*.

In RQ 2, we studied the effect of the problems on learning. There was a moderate positive correlation ($0.29 \leq r_s \leq 0.58$) between the amount of learning and problem solving contribution for most of the topics and both roles. Thus, with most of the topics the problems had a positive effect on learning. The correlations were strongest for *maintaining the motivation of the team members* ($r_s = 0.58$, $p = .002$ for managers) and *getting developers skilled with the implementation technologies* ($r_s = 0.55$, $p < .001$ for developers). On the other hand, for three topics there was practically no correlation for the managers. These topics were *gaining understanding of the features* and *quality requirements*, and *managing the versions of the source code*. For the developers, the weakest correlation was 0.17 ($p = .116$) related to *communicating within the team*. The correlations between learning and contributing to solving problems were stronger than those between learning and being affected by the problems meaning that the students learn more if they contribute to solving problems instead of just being affected by them.

In RQ 3, we studied how much the students learned of each topic. A vast majority of the students (84%) learned a lot (Mdn $\geq$ 6) about at least some of the studied topics. When considering all the topics as a whole, learning was moderate for both the managers (Mdn = 5.0) and developers (Mdn = 4.0). The difference between the roles is statistically significant ($p < .01$). Despite of the varying effect of the problems on learning among the topics, the median amount of learning varied only minimally among the topics for each role. The managers learned slightly more than the developers from almost all the studied topics. The developers learned more only about *getting developers skilled with the implementation technologies*, and both roles learned the same amount about *managing the versions of the source code*.

The amount of learning varied a lot among the students for each topic. For both roles, the interquartile range is 2.0–3.0 units for most of the topics. The amount of learning varied a lot also among the teams. For many topics there was a team or teams where the majority of the team members learned a lot ($6.0 \le \text{Mdn} \le 7.0$), but also teams where the majority of the team members learned only little ($2.0 \le \text{Mdn} \le 3.0$). Finally, the amount of learning varied also among the team members even among the students in the same role. The students' previous expertise with each topic was rather low ($\text{Mdn} = 3.0$) and did not affect the amount of learning for most of the topics.

In RQ 4, we studied the effect of problems on customer satisfaction based on the correlations between encountering problems and customer satisfaction. Customer satisfaction suffered most when problems were encountered related to *managing which tasks to do next and how* (developers: $r_s = -0.84$, $p < .001$; managers: $r_s = -0.66$, $p = .008$). Problems related to *communicating with the customer* decreased customer satisfaction also a lot, but only when the developers encountered them ($r_s = -0.68$, $p = .005$). Problems related to *managing customer's expectations* decreased customer satisfaction moderately both when the developers ($r_s = -0.47$, $p = .075$) and the managers ($r_s = -0.52$, $p = .046$) encountered them. For most of the other topics, the correlations are weak meaning that the problems with those topics did not have a large negative effect on customer satisfaction.

Our results help the teachers of similar capstone project courses make informed decisions on whether to increase or decrease problems with a particular topic. From the cost-benefit point of view, the best topics to struggle with problems are *effort estimation, testing*, and *technology skills.* They are educational for both roles and do not decrease customer satisfaction more than a little at most. Furthermore, problems related to *quality requirements* are educational to the developers and do not decrease customer satisfaction. Problems related to *customer communication* are educational to both roles, but if the managers struggle with them, customer satisfaction decreases a lot. The worst case from the cost-benefit point of view is when the developers encounter problems with *task management*, because then customer satisfaction decreases very much and the problems are not even educational.

Further studies are needed to identify factors that cause the differences in learning on the individual and team level. Then those factors could be taken into account in course designs in order to maximize learning for all the students. Relevant factors could include, e.g., 1) various project characteristics (customer, problem domain), 2) the detailed responsibilities and interests of the students that define the activities they focus on, and 3) the amount and type of mentoring during the project. Further studies could also try to understand why the effect of problems on learning varies among the topics. For example, can it be that some topics, such as understanding requirements, require careful attention in most of the projects even if no explicit problems were encountered with the topic, but some other topics, such as maintaining team's motivation, can be ignored without negative consequences in some teams.

References

ACM/IEEE CS Joint Task Force on Computing Curricula, 2015. Software Engineering 2014—Curriculum Guidelines For Undergraduate Degree Programs in Software Engineering. IEEE-CS & ACM 2015.

Ahtee, T., Poranen, T., 2009. Risks in students' software projects. In: 22nd Conference on Software Engineering Education and Training, pp. 154–157.

Broman, D., Sandahl, K., Abu Baker, M., 2012. The company approach to software engineering project courses. IEEE Trans. Educ. 55 (4), 445–452.

Bruegge, B., Krusche, S., Alperowitz, L., 2015. Software engineering project courses with industrial clients. ACM Trans. Comput. Educ. 15 (4), 17.

Clark, N., 2005. Evaluating student teams developing unique industry projects. In: 7th Australasian Conference on Computing Education, pp. 21–30.

Dawson, R., 2000. Twenty dirty tricks to train software engineers. In: 22nd International Conference on Software Engineering, pp. 209–218.

Dugan, R.F., 2011. A survey of computer science capstone course literature. Comput. Sci. Educ. 21 (3), 201–267.

Fincher, S., Petre, M., Clark, M., 2001. Computer Science Project Work Principles and Pragmatics. Springer-Verlag.

Foddy, W., 1993. Constructing Questions for Interviews and Questionnaires: Theory and Practice in Social Research. Cambridge University Press, Cambridge, UK.

Goold, A., 2003. Providing process for projects in capstone courses. In: 8th Annual Conference on Innovation and Technology in Computer Science Education, pp. 26–29.

IBM Corp., 2015. IBM SPSS Statistics for Windows, Version 23.0. IBM Corp., Armonk, NY.

Koolmanojwong, S., Boehm, B., 2013. A look at software engineering risks in a team project course. In: 26th Conference on Software Engineering Education and Training, pp. 21–30.

Mahnic, V., 2012. A capstone course on agile software development using scrum. IEEE Trans. Educ. 55 (1), 99–106.

Pournaghshband, H., 1990. The students' problems in courses with team projects. SIGCSE Bull. 22 (1), 44–47.

Pyster, A., 2009. Graduate Software Engineering 2009 (GSwE2009): curriculum guidelines for graduate degree programs in software engineering. Integrated Software & Systems Engineering Curriculum Project. Stevens Institute of Technology.

Runeson, P, Höst, M., 2009. Guidelines for conducting and reporting case study research in software engineering. Empir. Softw. Eng. 14 (2), 131–164.

Stehle, S., Spinath, B., Kadmon, M., 2012. Measuring teaching effectiveness: correspondence between students' evaluations of teaching and different measures of student learning. Res. High. Educ. 53 (8), 888–904.

Todd, R.H., Magleby, S.P., Sorensen, C.D., Swan, B.R., Anthony, D.K., 1995. A survey of capstone engineering courses in North America. J. Eng. Educ. 84 (2), 165–174.

Vanhanen, J., Lehtinen, T.O.A., 2014. Software engineering problems encountered by capstone project teams. Int. J. Eng. Educ. 30 (6A), 1461–1475.

Zell, E., Krizan, Z., 2014. Do people have insight into their abilities? A metasynthesis. Persp. Psychol. Sci. 9 (2), 111–125.

Dr. Jari Vanhanen is a university lecturer in software engineering with Aalto University, and has been the responsible teacher for the capstone software development project course at Aalto University since 2001. He has a D.Sc. degree from Aalto University.

Dr. Timo O. A. Lehtinen is a science adviser at the Academy of Finland. He is a former postdoctoral researcher at Aalto University and his research work has focused on software project retrospective methodologies and outcome. He has a D.Sc. degree from Aalto University.

Dr. Casper Lassenius is an associate professor at Aalto University. His current research interests include agile and lean software development, global software engineering, and software quality assurance. He has a D.Sc. degree from Aalto University.