Monitoring Software Development Teams in the Academia

Maíra Marques, Sergio F. Ochoa
Computer Science Department, Universidad de Chile
{mmarques, sochoa}@dcc.uchile.cl

Abstract

Performing real world software projects in the academia helps students understand what they will face in the industry. These experiences also contribute to realize the challenges of working collaboratively in an effective and efficient way. This article hypothesizes that a formative weekly monitoring of development teams in an academic scenario helps students to work coordinately and also to be more productive and effective. Trying to validate these hypotheses, we have defined and used a Formative Weekly Monitoring (FWM) method. Five development teams were monitored during twelve weeks using this method, and the results were compared to those obtained by seven non-monitored teams that worked in the same development scenario. This comparison of results allows us to preliminary conclude that the formative weekly monitoring makes development teams more effective and coordinated, but not more productive.

1. Introduction

Computer science programs strive to prepare future software engineers for industry work, promoting core-computing concepts that will allow students to become lifelong learners, able to keep pace with innovations in the discipline [1]. The approaches to teach technical knowledge have been driven by the advances in the software industry, and they have been well supported by the universities. However, the development of transversal capabilities such as coordination, decision-making and teamwork is usually less well supported in these programs [1]. These capabilities, also known as 'soft skills', usually impact the productivity, efficiency and efficacy of software development teams. It comes as quite a shock when novice software engineers enter the professional engineering workforce and realize that these transversal capabilities are highly important for the industry [2] [3].

Conscious of that situation, the academia tries to create recipes to support the development of these capabilities with future software engineers, but unfortunately this is not an easy task. Several researchers indicate that this challenge can be faced with software development courses that imitate the industry activities [4] [5] [6] [7]. These kinds of courses usually help address the problem, but they require extensive effort from instructors and teaching assistants, just to produce a small improvement in the student's capabilities. Some researchers state that this lack of soft skills is a consequence of the apprentice attitude that is usually adopted by the students while they are in the academia, instead of acting as software professionals [8] [9]. What is clear is that there is not a "silver bullet" to solve this problem.

In this research work the authors hypothesize that performing a formative weekly monitoring of software development teams (H1) positively impacts the coordination among team members, (H2) increases the amount of software built by the team, and (H3) makes teams more effective. These hypotheses are based on empirical results obtained by the authors in a previous work, when monitoring software development teams in the academia [10]. In order to validate these hypotheses we have defined a simple monitoring method named *Formative Weekly Monitoring* (FWM) method that adheres to the definition of Marks et al. [11], which indicates that it is "a process in the action phase category that involves observing the actions of teammates, watching for performance discrepancies, and providing feedback and assistance to those in need". Based on the theory of formative assessment, the FWM can be considered "formative" because its main purpose is to contribute to improve students' capabilities [12]. The proposed monitoring method involves watching and listening to team members during short interviews, and giving them feedback when needed.

The FWM method was used to monitor five development teams during twelve-weeks projects. The obtained results were compared to those obtained by non-monitored groups that developed also a software product during twelve weeks. This comparison showed that the monitored teams were more effective and coordinated, but not more productive.

Next section describes the instruments and strategies used by the Academia to help students to be more effective and productive in their developments. Section 3 describes the FWM method. Section 4 presents the experimentation scenario. Section 5 explains the methodology used in this study. Section 6 shows and discusses the obtained results. Section 7 presents the conclusions and the future work.

2. Background

According to Jong and Elfring [13] team monitoring reduces motivational losses and increases the likelihood of detecting free riding and "social loafing". This helps focus a team's efforts on reaching the team goals over the individual interests [14]. Monitoring can operate as an implicit coordination mechanism [15] by making team members more aware of the actions, timing, and performance of others. Thus, people increase their ability to synchronize their contributions in ways that maximize team goal attainment [16]. The effects of team monitoring can be compared with the effects of peer review, since the principle is the same: to make students work as they are supposed to. This is the most effective way to promote quality and productivity [17].

The use of team monitoring in software engineering courses is not a well-researched topic, though the use of team coaching is quite common. In team coaching, an experienced developer or instructor assists students' teams while they run a software process. The coach helps them to address internal issues (e.g. communication and coordination), mentoring students to use their knowledge and reviewing their deliverables [18]. Computer-mediated team monitoring is also used to help improve the coordination and feedback processes, and the team performance [16].

This work explores the impact of using a formative weekly monitoring in software teams in the academia. This type of monitoring can be understood as an informal formative assessment. This assessment involves qualitative feedback (rather than scores) and focuses on the details of content and students performance [19]. This concept can be best understood by a quote from Stevens et al. [20]: "When the cook tastes the soup, that is formative". In our case, we use a formative weekly monitoring to see if it affects the coordination, effectiveness and amount of software built by software teams in the academia. Next section introduces the proposed FWM method.

3. Formative weekly monitoring method

This method is based on the idea of Marks et al. [11], and it was designed to monitor the team members performance in the undergraduate course CC5401 (Software Engineering II), taught in the Computer Science Department of the University of Chile. This method was initially designed to identify the students' performance observing their actions, watching for discrepancies among team members, and providing feedback when needed. However, after using it in practice we identified that it could help teams to improve their performance.

The monitored course involves two weekly lectures and one auxiliary class. Students that take this course usually are in the 10th semester of the program. The instructor groups the students in teams of 5 to 7 members, and they have to run a software project during 12 weeks. The projects involve a real client that interacts directly with the students. All projects have a set of deadlines, which are defined by the instructor when he assigns them to the teams. The projects goal, size and complexity have been previously agreed between each client and the instructor. The instructor (who has an important experience as software developer) is in charge of ensuring that every project can be done by the assigned team in twelve weeks. Typically the project goals are to solve an organizational problem through the development of a software solution. That solution should be put into production to the end of the project.

During classes the instructor remarks that the auxiliary classes are mandatory for at least three members of each team, and that every team will be interviewed (i.e. monitored) at the end of each auxiliary class. This monitoring does not impose penalties to team members in case of absences. During the sessions the monitor (who is not the teaching assistant or the instructor) observes the fulfillment of teammates assignments, the team members behavior, and the feasibility to reach the project deadlines, as suggested by Langfred [21]. However the focus of each monitoring session was mainly on this last issue.

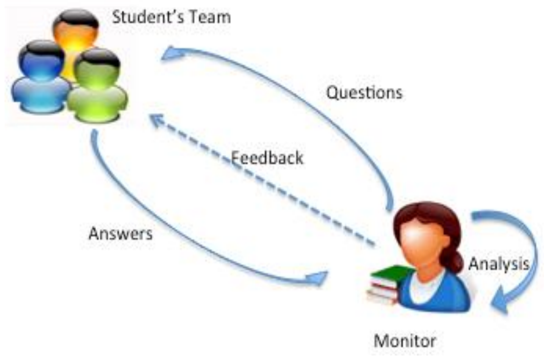


Figure 1. FWM method

Typically the monitor makes specific questions to teammates to determine the project status, and based on that, to help them discover weaknesses and risks in the work they are preforming (Fig. 1). In few cases, the monitor provides feedback and assistance about how to address these issues. Typically, the monitor is a person with professional experience in software development. The monitoring sessions were recorded and a transcription was made based on them.

The monitoring session starts with two simple questions such as: “How was the week?” and “Did everybody do what they were supposed to do?”. These questions are used to get some general feedback about the project evolution. After that the monitor focuses just on two elements: *risks identification* (i.e. the potential problems affecting the team or the project) and the actions to be made by the team to advance the project towards its final goal.

During these sessions the monitor ensures that every member can give their opinion. The monitoring questions intend to determine if team members are really conscious about the real problems that they have to address, their importance and timing, and the involved deadlines. The feedback that the monitor provides should allow the team members to make their own decisions; and they usually they are more general hints than direct advices. The target is not to coach the team members, but just to play the “psychologist” with them, making them realizes on their own where and how they are on their projects.

4. Experimentation scenario

The last two software engineering courses in the computer science program at University of Chile are Software Engineering II (CC5401) and Software Project (CC5402). During the last twelve years these courses have involved the development of real software projects, as a way to help transfer soft skills to undergraduate students. These projects differ in the strategy used to address them (e.g. the first one uses a structured process and the second uses an agile method), however both follow the same goal, and have a similar duration and development team size.

The course CC5401 was chosen to be monitored because their teams usually have major problems in *coordination*, *effectiveness* and *productivity*, which are the variables involved in the stated hypotheses. For this study we have monitored teams during two semesters: Spring 2010 (two teams) and Autumn 2012 (three teams). Their performance (in term of these variables) was compared to the one shown by non-monitored teams that participated in the semesters: Autumn 2011 (five teams) and Spring 2011 (two teams). The four semesters considered in this study are structurally comparable.

5. Methodology

The periodicity of the monitoring was weekly because the projects are short, and therefore the student teams do not have much time to waste. The weekly monitoring would give them the opportunity to refocus the project on what is important without loosing too much time out of track. During the sessions the monitor helped students find the answers by their own to the project questions, and also to deal with some teammates' disruptive behavior, such as the free riding and "social loafing".

In order to assess the students' soft skills acquisition, a formal peer-review was conducted using a survey. Such a survey has been used and evolved in this the course during the last seven years, and it is focused on determining how well each person fits with the behavior expected from a teammate. The students were assured that all opinions would be anonymous and that there would be no formal grades given based on that peer-reviewing process.

The instrument used to perform the peer-reviewing process considers eight items that should be rated by team members to evaluate each teammate. The rates were indicated using a five-point Likert scale (from 'strongly disagree' to 'strongly agree') [22]. The evaluated items were the following:

1. He/she assumes the project as a team effort, providing support in project tasks.
2. He/she is able to ask for help when problems are found.
3. He/she fulfills his/her tasks properly, working transparently and generating the most value out of each working day.
4. He/she demonstrates initiative to achieve project success.
5. He/she shows a communicative attitude facilitating the teamwork.
6. He/she has maintained good communication with client, generating value to project execution.
7. He/she demonstrates interest in improving performance on the execution of his/her activities and role within the project.
8. He/she is able to admit to mistakes and accept criticism.

To measure the reliability (internal consistency) of the peer-review the Cronbach's Alpha was used. The internal consistency describes the extent to which all the items in a test measure the same concept or construct and hence it is connected to the inter-relatedness of the items within the test [23].

Given the nature of the collected data, a research approach inspired in the grounded theory was chosen to conduct this study. According to Glaser and Strauss [24] grounded theory is the discovery of theory from data systematically obtained from social research. The aim of it is to generate or discover a theory.

There is a consistent body of literature that supports the use of grounded theory to study social phenomena in software engineering [25]. It uses a systematic process that begins with the data collection, and where the process stages can be overlapped. According to Glaser and Strauss [24], these stages are the following: Identify your substantive area (Software Engineering Education); Collect Data (Monitoring of software development projects); Open Code (Weekly monitoring of the teams); Memo Writings (During the whole monitoring notes were taken); Theoretical Sampling (The first semester of monitoring acted as sampling); and Theoretical Code (Hypotheses were created).

6. Obtained results

In order to evaluate the first hypothesis proposed (*H1- weekly monitoring of software development projects positively impacts the team members coordination*), a comparison between the last four semesters of the course CC5401 was done (Figure 2). Team coordination was determined using just the results of the peer-reviewing process, because they represent the teammates opinion, which probably are more accurate than the monitor opinion. The results obtained for the monitored teams were compared to the monitor opinion, and both of them were completely aligned.

Figure 2 shows the obtained results for these items. The results indicate that the teams work better when they are monitored weekly. Particularly, the team members' commitment (item 1) and initiative (items 2, 4 and 7) seem to improve considerably when the teams are monitored. The rest of the evaluated items also improve in monitored teams. These results (at least preliminarily) are aligned to the hypothesis H1.

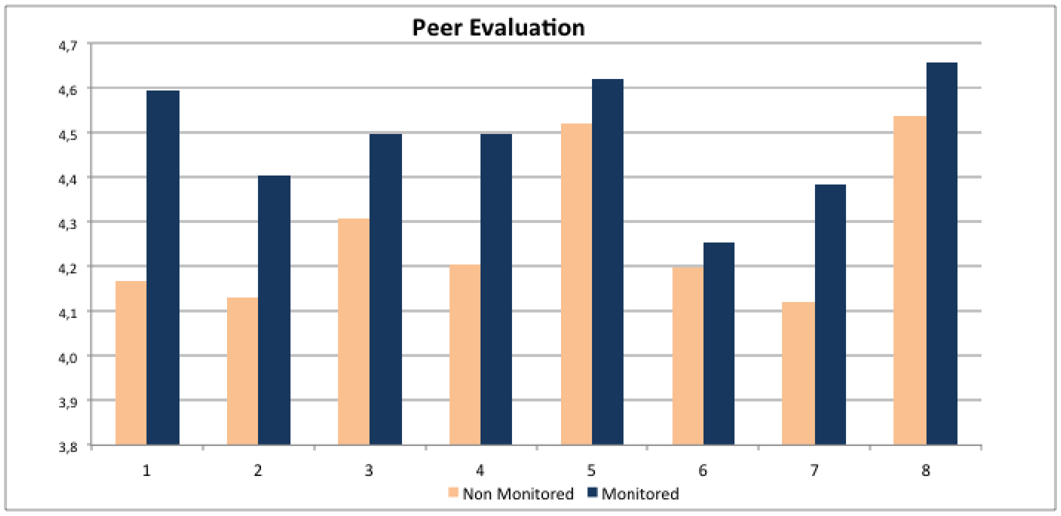


Figure 2. Teams comparison based on the peer-reviewing process

In order to analyze the amount of software built by these teams, we have used the software requirements involved in those projects. Every team in the course has to use a requirements management tool that was particularly developed to support the students and instructor in the project tracking. Such a tool keeps a record of the project requirements and their evolution. For each requirement the system stores its definition, accomplishment, priority and criticality. Using this information we determined the amount of software developed by each team.

Due the differences of effort required to accomplish with the variety of requirements considered in a project, we used function points (FP) [26] and a conversion table that allows determining, quite accurately, how many FPs are involved in a particular requirement. The conversion table (Table 1) was developed some years ago using information of small software companies from Chile.

Table 1. FP Conversion table

Component	FP	Description
CRUD	5	Create, read, update and delete functions of storage
Navigation	7	Navigation panel
Queries	2	Queries made to the database
Printed Reports	2	Formatted reports printed
Background Processes	8	Any background processes that is needed to be built
Notification	2	Email, message or any notification of events
Registration Processes	3	Login, password and user registration processes

The quality requirements involved in the projects were not considered in this analysis due they are comparable in every project. In the course every software product should show a certain level of maintainability, usability, performance, security and independence of the Web browsers. The way to measure these aspects are specified, and they are addressed by the students; therefore we can assume that the analyzed projects are comparable in terms of product quality. Among these projects there was just one project that had an extra quality requirement, which was portability. Due the team was not able to address such a requirement, the project was considered as comparable.

The function points assigned to each requirement for each project were estimated by two experienced software engineers, using Table 1 as reference. The values were then validated through the estimation done by the authors. There was a difference of 14% between the estimations of both groups, which indicates that it was coincidence about the effort required to accomplish with the projects requirements. Figure 3 shows the obtained results.

These results show that monitored teams have addressed a more important number of requirements than the non-monitored teams. These results would be supporting the Hypothesis 2: *weekly monitoring of software development projects increases team the amount of software built by the team*. However the difference between both groups is so small that we have to consider them as similar. Trying to identify if the number of team members makes a difference in the amount of software implemented, we have calculated the average number of FPs addressed by each team member. Figure 4 shows these obtained results, which are similar to those presented in Fig. 3. Therefore, based on this preliminary evidence, the second hypothesis seems to be false.

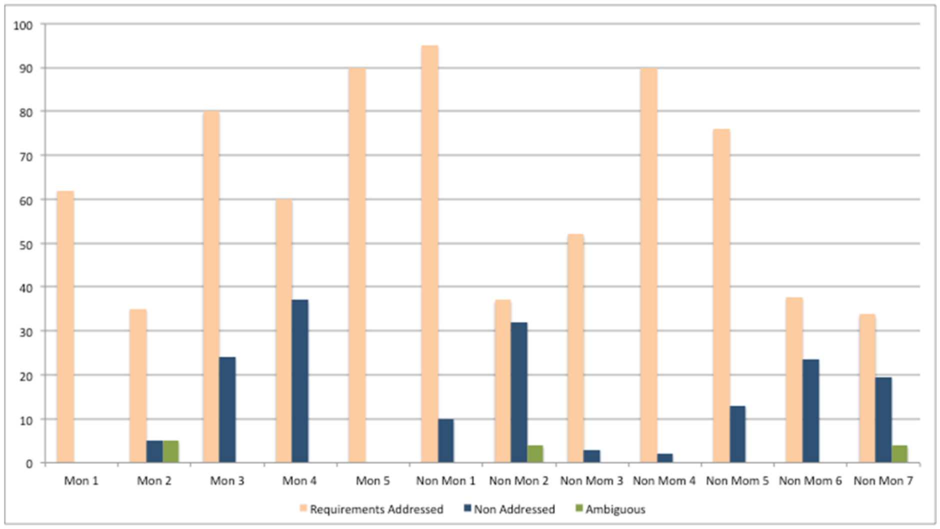


Figure 3. Number of FPs addressed by team

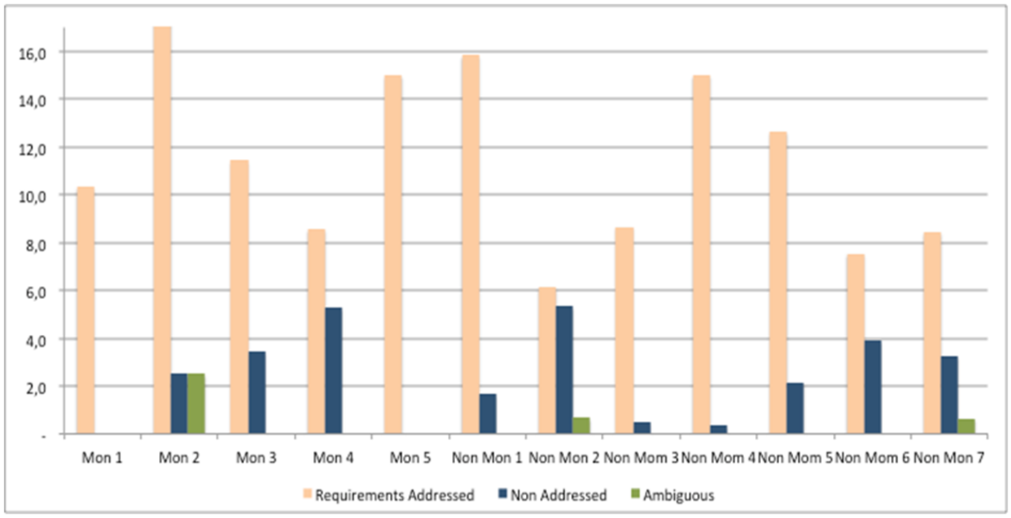
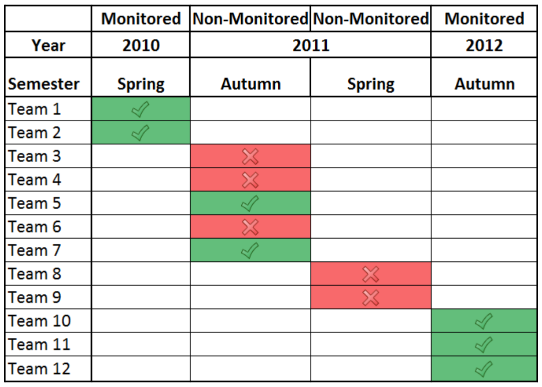


Figure 4. Average Number of FPs Addressed by Team Member

In order to analyze the effectiveness of the teams (H3 - *weekly monitoring of software development projects makes teams more effective*) we took into consideration each project outcome. Figure 5 shows the outcomes of the projects done in the course CC5401, considering monitored and non-monitored teams. Provided that the teaching team and the course content and dynamics did not change during the observed semesters, we can consider these teams as comparable.

The green cells correspond to successful projects; i.e. those that were able to put into production a software product that is useful for the client. Projects that did not reached such a goal were considered unsuccessful, and they were indicated with a cross and a red cell. The results show that monitored teams had always a positive outcome; however in the non-monitored ones there were both outcomes (i.e. positive and negative).



	Monitored	Non-Monitored	Non-Monitored	Monitored
Year	2010	2011		2012
Semester	Spring	Autumn	Spring	Autumn
Team 1	✓			
Team 2	✓			
Team 3		✗		
Team 4		✗		
Team 5		✓		
Team 6		✗		
Team 7		✓		
Team 8			✗	
Team 9			✗	
Team 10				✓
Team 11				✓
Team 12				✓

Figure 5. Projects outcomes

Trying to understand the reason that generates this difference of performance, we analyzed the requirements that were non-addressed by the teams. For this analysis we used the level of criticality that the team members assigned to each requirement. The tool used by the students to define the requirements allows three possible values for the requirements criticality: critical (i.e. mandatory), desirable (i.e. addressable if all mandatory requirements were addressed) or optional (i.e. addressable if we have time).

Figure 6 shows that the monitored teams were able to address all the critical requirements and the non-monitored teams were not able to do so. Addressing the critical requirements made the monitored teams much more effective, allowing them to put their product into production. Moreover it seems that the monitoring sessions allow members of those teams to focus on the critical (or important) aspects of the project instead of wasting time in non-mandatory issues, and that made the difference. These preliminary results are aligned with the Hypothesis 3.

In order to verify this finding we did the same analysis for the Software Project course (CC5402), which is similar to CC5401 and it involves also weekly monitored projects. Figure 7 shows the obtained results, which are completely aligned to those obtained in projects of the CC5401 course. These results reinforce the idea that the Hypothesis 3 would be probably true. The main reason for this effectiveness improvement is because the monitored teams are more focused on the important aspects of the project.

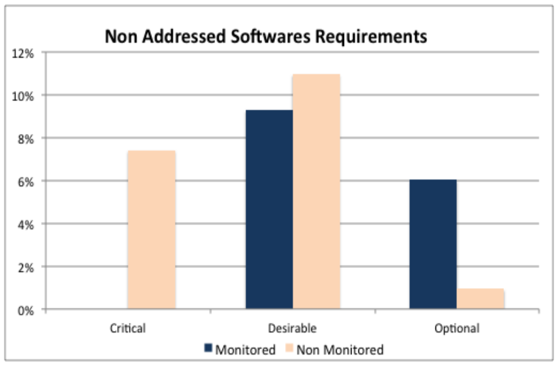
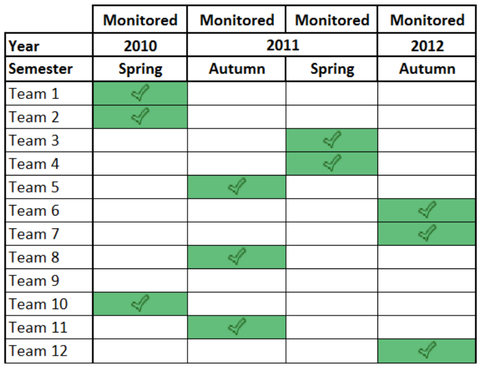


Figure 6. Non-Addressed software requirements



	Monitored	Monitored	Monitored	Monitored
Year	2010	2011		2012
Semester	Spring	Autumn	Spring	Autumn
Team 1	✓			
Team 2	✓			
Team 3			✓	
Team 4			✓	
Team 5		✓		
Team 6				✓
Team 7				✓
Team 8		✓		
Team 9				
Team 10	✓			
Team 11		✓		
Team 12				✓

Figure 7. Project outcomes in the CC5402 course

In order to ensure that the data source used in this study is valid, we have calculated the Cronbach's Alpha coefficient for the eight questions from the peer-review form, for the 58 respondents (from a total of 67 participants). Table 2 shows the mean and the standard deviation of the results. The closer Cronbach's Alpha coefficient is to 1.0 the greater the internal consistency of the items in the scale. The average of Cronbach's Alpha calculated was 0.8. According to Gliem and Gliem [27] a value of Cronbach's Alpha of 0.8 from Likert-type scales is good and it is considered a reasonable goal.

Table 2. Mean and standard deviation of the peer-reviewing questions

Questions	Mean	Standard Deviation
He/she assumes the project as a team effort, providing support in project tasks.	4.39	0.96
He/she is able to ask for help when problems are found.	4.35	1.02
He/she fulfills tasks properly, working transparently and generating the most value out of each working day.	4.45	0.91
He/she demonstrates initiative to achieve the project success.	4.38	0.98
He/she shows a communicative attitude facilitating the teamwork.	4.57	0.86
He/she has maintained good communication with the client, generating value to project execution.	4.31	1.04
He/she demonstrates interest in improving his/her performance on the execution of activities and role within the project.	4.35	0.99
He/she is able to admit to mistakes and accept criticism	4.67	0.78

7. Conclusions and future work

This research work explores the potential impact that the formative weekly monitoring (FWM) has on software development teams in the academia. In order to do that this article presents a study of twelve projects performed in a tenth-semester software engineering course that is delivered in the computer science undergraduate program at the Universidad de Chile.

Three hypotheses were defined for this study: *the FWM of software development teams (H1) positively impacts the coordination among team members, (H2) increases the amount of software built by the team, and (H3) makes teams more effective.*

The preliminary results are statistically significant and they indicate that the FWM makes teams more effective (H3). In that sense, the monitored teams had a project success rate of 100%, and the non-monitored ones succeeded in the 29% of the cases. Moreover, the FWM also positively impacts the team coordination (H1), and it helps the team to stay focused on what is relevant. If the team knows what is relevant, the coordination of the individual efforts will be smoother than ever. Unfortunately, this monitoring seems to have no impact on the amount of software built by the monitored teams, compared to the software production of non-monitored teams (H2).

Having positive outcomes in software projects performed in the academia is something that does not happen everyday, but that is highly important for the future software engineers. The FWM seems to be a valid instrument to help students to reach such a goal and learn to work in teams in real-world projects. The FWM can help team members put their effort on what really matters, minimizing the internal friction among team members. It let students know they are not drifting away, giving them confidence on what they are doing. The idea that the students will have to report in what they did, no matter that was an informal report, makes students more aware of their own work; and therefore they can work in a more coordinated and effective way.

A more extensive analysis of how and why the weekly monitoring helps teams to be more effective will be done as part of the future work. It is also worth evaluating why the positive outcomes of the course CC5402 are so much higher than the numbers shown by the software industry. According to the 2009 CHAOS Report [28], the project success rate in the software industry is approximately 32%.

Acknowledgement

The work of Maíra Marques Samary was supported by the PhD Scholarship Program of Conicyt Chile (CONICYT-PCHA/Doctorado Nacional/2012-21120544).

8. References

[1] Begel, A., and B. Simon. "Novice Software Developers, All Over Again." Proceedings of the 4th Workshop on Computing Education Research. ACM Press (2008): 3-14.

[2] Perlow, L. "The Time Famine: Toward a Sociology of Work Time." Administrative Science Quarterly Vol. 44, no. 1 (1999): 57-81.

[3] Tomayko, J. E., and O. Hazzan. "Human Aspects of Software Engineering." Hingham, MA: Charles River Media, 2004.

[4] Tvedt, J. D., R. Tesoriero, and K. A. Gary. "The Software Factory: Combining Undergraduate Computer Science and Software Engineering Education." Proceedings of the 23rd International Conference on Software Engineering. IEEE Press (2001): 633-642.

[5] Gehrke, M., et al. "Reporting About Industrial Strength Software Engineering Courses for Undergraduates." Proceeding of the 24th International Conference on Software Engineering. ACM Press (2002): 395-405.

[6] Robillard, P., P. Kruchten, and P. D'Astous. "Yoopeedoo (UPEDU): a Process for Teaching Software Process." Proceedings of the Software Engineering Education and Training. IEEE Press, (2001): 18-26.

[7] Rico, D. F., and H. H. Sayani. "Use of Agile Methods in Software Engineering Education." Proceedings of the Agile Conference. IEEE Press, (2009) 174-179.

[8] Goldratt, E. M. "Critical Chain. MA: The North River Press, 1997.

[9] Boehm, B. W., and D. Port. "Educating Software Engineering Students to Manage Risk." Proceedings of the 23rd International Conference on Software Engineering. IEEE Press, (2001): 591-600.

[10] Marques, M., S. F. Ochoa, A. Quispe, L. Silvestre, and A. Villena. "Coordination and Pressing: A Formula for Teamwork - A Case Study." Proc. of the IEEE Int. Conf. on CSCWD. IEEE Press, (2010): 83-88.

[11] Marks, M. A., J. E. Mathieu, and S. J. Zaccaro. "A Temporally Based Framework and Taxonomy of Team Process." Academy of Management Review Vol. 26 (2001): 356-376.

[12] Darling-Hammond, A., and J. Bransford. "Preparing Teachers for a Changing World: What Teachers Should Learn and be Able to Do." Jossey-Bass, New York, 2007.

[13] De Jong, B. A., and T. Elfring. "How Does Trust Affect the Performance of Ongoing Teams? The Mediating Role of Reflexivity, Monitoring, and Effort." The Academy of Management J., Vol.53, no. 3 (2010): 535-549.

[14] Jones, G. R. "Task Visibility, Free Riding, and Shirking: Explaining the Effect of Structure and Technology on Employee Behavior." Academy of Management Review no. 9 (1984): 684.

[15] Rico, R., M. Sanchez-Manzanares, F. Gil, and C. Gibson. "Team Implicit Coordination Processes: A Team Knowledge Based Approach." Academy of Management Review no. 33 (2008): 163.

[16] Marks, M. A., and F. J. Panzer. "The Influence of Team Monitoring on Team Processes and Performance." Human Performance Vol. 17, no. 1 (2004): 25-41.

[17] Garousi, V. "Applying Peer Reviews in Software Engineering Education: An Experiment and Lessons Learned." IEEE Transactions on Education, Vol. 53, no. 2 (2010): 182-188.

[18] Bareiss, R., and M. Griss. "A Story-Centered, Learn-by-Doing Approach to Software Engineering Education." ACM SIGCSE Bulletin, Vol. 40, no. 1 (2008): 221-225.

[19] Scriven, M. The Methodology of Evaluation. AERA Monograph Series on Evaluation. Rand McNally, 1967.

[20] Stevens, F., F. Lawrenz, and L. M. Sharpe. User-Friendly Handbook for Project Evaluation: Science, Mathematics, Engineering and Technology Education". National Science Foundation, 1993.

[21] Langfred, C. W. "Too Much of a Good Thing? Negative Effects of High Trust and Individual Autonomy in Self-Managing Teams." Academy of Management Journal Vol. 47 (2004): 385-399.

[22] Likert, R. "A Technique for the Measurement of Attitudes." Archives of Psychology, 1932.

[23] Cronbach, L. "Coefficient alpha and the internal structure of tests." Psychomerika, no. 16 (1951): 297-334.

[24] Glaser, B. G., and A. L. Strauss. The Discovery of Grounded Theory: Strategies for Qualitative Research. New York: Aldine de Gruyter, 1967.

[25] Mai, U., D. Swift, T. Wiggins, and R. Luechtefeld. "A Grounded Theory Approach to Effects of Virtual Facilitation on Team Communication and the Development of Professional Skills." Proceedings of the Frontiers in Education Conference. IEEE Press (2011): T2C-1- 5.

[26] Albrecht, A. "Software Function, Source Lines of Code, and Development Effort Estimation - A Software Science Validation." IEEE Transactions on Software Engineering Vol.9, no. 6 (1983): 639-648.

[27] Gliem, J. A., and R. R. Gliem. "Calculating, Interpreting and Reporting Cronbach's Alpha Reliability Coefficient for Likert-Type Scales." Proceedings of the Midwest Research to Practice Conference in Adult, Continuing, and Community Education, (2003).

[28] The Standish Group. "CHAOS Summary Report". 2009.