SIGCSE: G: Monitoring - An Intervention to Improve Team Results in Software Engineering Education

Maíra Marques
Universidad de Chile
Beauchef 851
Santiago, Chile
mmarques@dcc.uchile.cl

ABSTRACT

Software engineering education is currently being taught in many universities with a hands-on approach. Students develop software in teams, intending to simulate industry, in order to minimize the gap between what universities teach and what industry needs. The industry "simulation" that a university performs in software engineering courses is a good approach but it is normally undermined by student bad behaviors (free-riding, social loafing and student syndrome). In this work I propose the use of reflexive weekly monitoring (RWM) to mitigate these students' bad behaviors that can disrupt team results and learning. The use of the RWM in the past five semesters in a software engineering course, has improved teams' results in terms of both, project deployment rate and final grades.

Keywords

software engineering, software engineering education, monitoring, reflexive monitoring, teamwork self-regulation and learning

1. PROBLEM AND MOTIVATION

Working in teams is not easy and it is one of the biggest issues faced on software engineering education [6][9][11][17]. As stated by Radermacher et al.[17], there exists a gap between students abilities and industry expectation. One of the aspects they report is a "lack of teamwork skills and collaborative ability".

Approaches to teach technical knowledge have been driven by the advances in software industry, and they have been well supported by the universities. However, the development of skills and competences such as coordination, decision-making and teamwork is usually supported at a lower level in these programs [2]. These competences, also known as "soft skills", usually impact the productivity, efficiency and efficacy of software development teams.

Conscious of that situation, the academia has proposed different approaches to support the development of these skills in the 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 [7][8][18][21]. This kind of courses usually help to 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 the difficulty of developing these soft skills in students is due to an apprentice attitude instead of a professional one [3][4][19] that students adopt.

There are some threats to teamwork that are related to student behavior: free riding, social loafing and student syndrome. Free riders are students who do not contribute to the team [22]. Social loafing happens when team members do not do their fair share of work, i.e; they make a lower effort than they would in an individual project [5]. The student syndrome is the phenomenon where students do not fully apply themselves to the project until just before the deadline [10]. The consequences of these threats to successful teamwork is that they affect the team as a whole, and lead to lower team performance and lower team motivation [22]. The team effectiveness (ability of students to deliver a software that can be deployed and used by the client) is also undermined by the students syndrome [7].

Another aspect to take into consideration is that the nature of students' projects is usually different from real-world projects which are bounded by contract, and team members are accountable for their performance. In software engineering education there is no contract and students may be tempted of not doing what they are expected, without harsh consequences [6]. Threats of course failure or bad grades may not be enough for students to change their behavior.

2. BACKGROUND AND RELATED WORK

According to Duim et al. [22] the free riders problem occurs when students know that instructors do not have a real insight about each team member's effort. So, they proposed a solution that reduces the anonymity of students' work through time logging, continuous evaluation (monitoring) and a pre-defined process. Hazzan [12] showed that team rewards and individual rewards can be complimentary for boosting cooperation.

Trytten [20] suggested applying peer evaluation, where part of students' grades depend on other team members evaluation of the work done by the individual. Yu [23] states that the team size can also be an issue, the larger the

SIGCSE '16 March 02-05, 2016, Memphis, TN, USA

ACM ISBN 978-1-4503-3685-7/16/03.

DOI: http://dx.doi.org/10.1145/2839509.2851054

team the higher the probability that students syndrome occurs. Johns-Boast and Flint [14] used continuous monitoring (through observation and peer assessment) and evaluation to minimize the students syndrome.

Borrego et al. [5] conducted a literature review on the effectiveness of teams in engineering students projects. They considered social loafing and free riding problems as motivational issues. They report finding four pedagogical recommendations to avoid these issues: compelling project with inherent value, peer evaluation of individual effort, small teams and complex tasks. They considered the student syndrome and interdependence, i.e., that the team as a whole will complete the work as committed. In order to promote higher interdependence and lower student syndrome they make three pedagogical recommendations: complex projects, group processing (includes a low monitoring process) and group gradings.

Since different types of monitoring were already used to deal with student behavior (different team contexts); we used in this work the definition of monitoring given by Marks et al. [15], who define it as "a process in the action phase category that involves observing the actions of team members, watching for performance discrepancies, and providing feedback and assistance to those in need".

3. APPROACH AND NOVELTY

To deal with these challenges (student behavior issues and students projects not being bounded by contracts) I propose the Reflexive Weekly Monitoring (RWM), which consists of a monitor weekly meeting with the team. The monitor is neither a team member, nor a Scrum master; he is responsible of rising warnings to students in case they are needed; he intends to guarantee that all team members are committed to the project success as equally as possible. During the sessions the monitor, who is not the teaching assistant or the instructor and typically is a person with professional experience in software development; observes the fulfillment of team assignments, the team members behavior, and the feasibility of reaching the project deadlines. However, the focus of each monitoring session is mainly on this last issue.

The RWM sessions involve three sequential steps (Figure 1): the *monitor diagnosis*, the team *reflection* and the *closure*. In each session, the monitor asks specific questions following a script according to the course schedule in order to understand the current status of the project and the team. The questions are simple and involve direct answers. For instance: Has everybody accomplished the assigned tasks? Is your project on schedule? Are you going to be able to accomplish the deadlines? Is your team working well? The monitor script covers most categories involved in Jacobson's approach [13]: opportunity, stakeholders, requirements, software system, work, team, and way of working.

Typically the monitor asks specific questions to team members to evaluate the project status, and based on that, to help them discover weaknesses and risks in the work they are preforming. In a few cases, the monitor provides feedback and assistance about how to address these issues.

During the monitoring session the monitor ensures that every member can give his or her opinion. The monitoring questions intend to determine if team members are really conscious about the actual problems that they have to address, their importance and timing, and the involved deadlines. The feedback that the monitor provides should allow

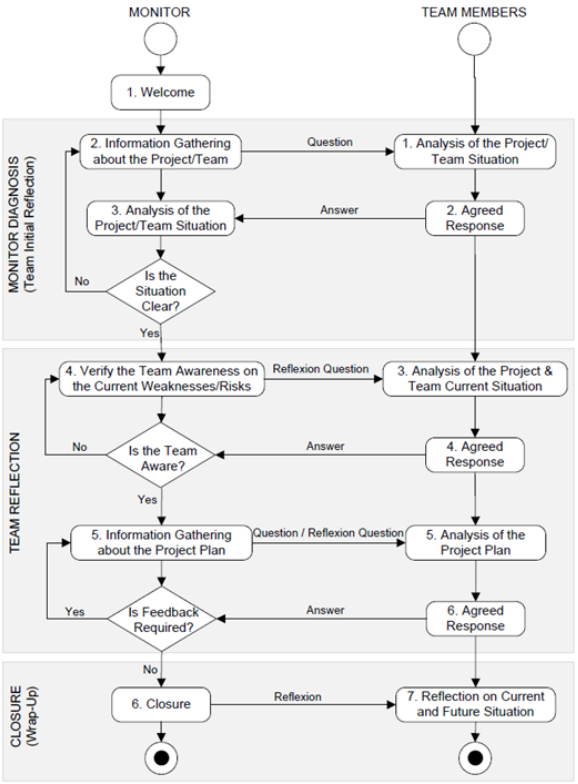


Figure 1: Structure of a RWM Session.

team members to make their own decisions; and provide general hints more than direct advices. The goal is not to coach team members, but just to play the "psychologist" with them, making them realize by themselves where and how they are on their project. In Figure 1 we present the structure of the RWM.

- *Monitor Diagnosis* - The monitoring session starts with two simple questions: "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 status. After that, the monitor focuses on one elements: analysis of the project/team situation (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.

- *Reflection* - According to the Diagnosis step the monitor check if team members are aware of project situation (possible project risks) and in line with that the monitor make concrete questions instilling them to think, analyze and reflect about it (i.e. a team member reports that there is some conflict in a decision about the design of something in the project; the monitor asked questions "What are the advantages of each possibility?"; "What are the costs to the project in terms of time spent with each possibility?" The monitor in these specific case suggested to students to do a short list of pros and cons; and students have to think and evaluate the situation and the possibilities). As a result the team have to agree (after reflecting) with a strategy to solve the problem/situation.

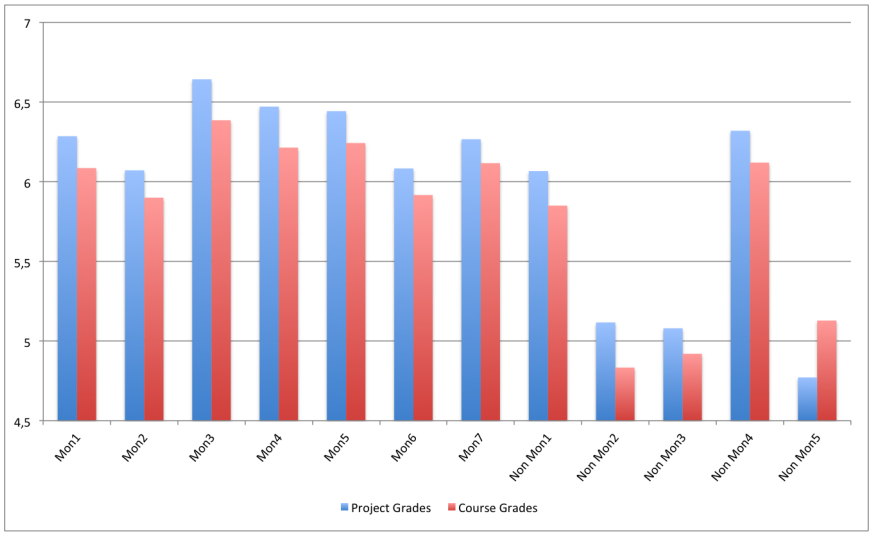


Figure 2: Student Evaluations

- *Closure* - In the closure the monitor asks students about the next steps of the project to see if they are reasonable, and if the project is within schedule. The session ends with a wrap-up with the priority actions that the team needs to address within the next week.

4. EXPERIMENTATION SCENARIO

At the Universidad de Chile the Computer Science Engineering program lasts 11 semesters and has the major goal of addressing all the five curricula guidelines within ACM-IEEE [16] in the same program. Related to software engineering it has three courses, the first one is more theoretical than practical and the other two have a practical focus. The two practical courses are different; the first one; Software Engineering II - CC5401 uses a structured process, the development is made by students on their own time, and the development is completely decentralized. The second one Software Project - CC5402 uses an agile method and all the development is made on clients facilities, and all students work more or less together. However both follow the same goal, and have similar duration and development team size.

The course CC5401 was chosen to be monitored because its teams traditionally have major problems in coordination, effectiveness and productivity. For this study we have monitored teams during five semesters, where some teams were monitored and some were not according to teams schedule (random selection). The semesters considered in this study are structurally comparable: same instructor and same type of project.

More specifically, CC5401 is a mandatory course for fifth year students. Here students have to develop web information systems for real clients during the semester. Teams are formed of 5-7 students. Each student is assigned a specific role, and they have to follow an ad-hoc prescriptive software process. The project represents 60% of the final grade.

During classes the instructor remarks that the auxiliary classes are mandatory for at least four members of each team, and that the teams previously selected to be monitored will be monitored at the end of each auxiliary class.

5. RESULTS AND CONTRIBUTIONS

The RWM method was used to monitor seven out of twelve software development teams, composed by computer science undergraduate students enrolled in Software Engineering II. This experience involved 107 students grouped into 12 teams along five semesters.

5.1 Results

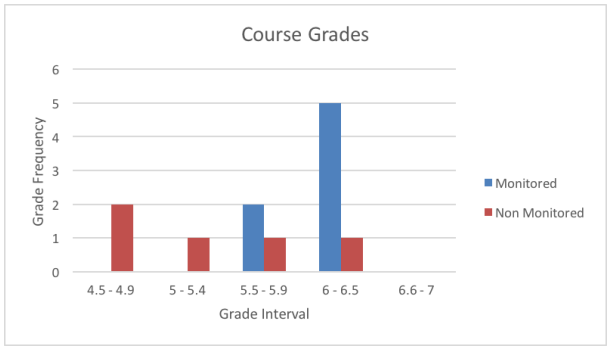
In order to evaluate RWM the final grades of all the teams were compared. These grades are in a scale from 1 to 7; and there is 1 point that is given as basis point (there are no grades lower then 1) and to be approved the student needs a grade higher than 4. During the five semesters twelve teams were observed; 7 were monitored and 5 were not.

The teaching team, course content and dynamics had not changed during the observed semesters, so they are comparable. Grades are mainly related to the quality of the product the client receives.

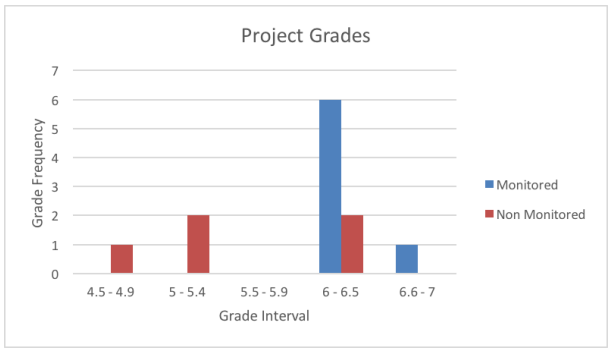
Figure 2 shows all the project and course grades; teams labeled *Mon* are those that were monitored using RWM and those labeled *Non Mon* were not monitored. Figure 3a shows the course grades frequency of monitored and non monitored teams and Figure 3b shows the project grades frequency. The average project and course grades of monitored teams are 6.32 and 6.12 while they are 5.47 and 5.37, respectively, for non-monitored teams. Project and courser grades for monitored teams were 19% and 17%, higher respectively.

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, degree of completion, priority and criticality. Using this information we determined the amount of software developed by each team. To evaluate the effort teams have to make to build their software we used a light version of function points (FP) [1].

To minimize a possible bias in the conversion of requirements to FP (due to its subjectivity), two experienced soft-



(a)



(b)

Figure 3: Students Evaluations (a) Course Grades and (b) Project Grades

ware engineers were asked to do the same conversion using the methodology. The values that the experienced software engineers estimated for each project were compared to the values previously estimated. In this comparison there was a difference of 14% in average (the biggest difference was of 18%, the lowest one was of 5% and the median was of almost 10%) between the estimations of both groups, which indicates that the amount of FP estimated by experts are similar.

Table 1 presents the teams and the amount of code developed per team member. The monitored teams were able to build in average 74.5 FP while the non monitored teams were able to achieve 70.8 FP; more or less the same amount of code was done by both monitored and non monitored teams.

However when we look the final results of each team regarding the software built, the results are not the same. In Table 2 it is possible to see the software outcome of each team. 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 reach such a goal were considered unsuccessful, and they were indicated with a cross and a red cell. The results show that monitored teams always had a positive outcome; however in the non-monitored ones there were both outcomes (i.e. positive and negative). All the monitored teams were able to deploy the software, 100% deployment rate and the non monitored ones two out of five were able, 40% deployment rate.

5.2 Contributions

Table 1: Team productivity

	Team	FP
Monitored	Team 1	62
	Team 2	87
	Team 3	80
	Team 4	60
	Team 5	90
	Team 6	52
	Team 7	91
Non monitored	Team 1	95
	Team 2	41
	Team 3	52
	Team 4	90
	Team 5	76

Table 2: Project Results (meaning: ✗ - not deployed, ✓- deployed)

Team	Monitored	Non-Monitored
Team 1	✓	–
Team 2	✓	–
Team 3	✓	–
Team 4	✓	–
Team 5	✓	–
Team 6	✓	–
Team 7	✓	–
Team 1	–	✗
Team 2	–	✓
Team 3	–	✓
Team 4	–	✗
Team 5	–	✗

This research work explores the potential impact that the RWM has on software development teams in the academia. In order to do that this article presents a study of twelve projects performed in a fifth-year software engineering course that is delivered in the computer science undergraduate program at the Universidad de Chile.

The preliminary results show that the reflexive weekly monitoring improve teams results in terms of course and project grades and in terms of software deployment.

Moreover, despite the fact that intuitively the RWM could positively impact the amount of work achieved by the team, looking at the data it is not possible to find any evidence of it. This means that monitored teams were able to do a similar amount of work as the non monitored teams, but they were able to have a better deployment rate. This means that the fact that they have to reflect on what they have to do, and on what was happening in the project with the monitor helped the teams to be more effective and more focused on doing what really matters for the client; it helped the teams to stay focused on what is relevant.

Having positive outcomes in software projects performed in the academia is not frequent, but it is highly important for the future software engineers. The RWM is a useful instrument to help students to reach such a goal and learn to work

in teams in real-world projects. The RWM 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 what they did, even as an informal report, makes them 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 much higher than the numbers shown by the software industry.

6. ACKNOWLEDGMENTS

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

7. REFERENCES

[1] A. J. Albrecht and J. E. Gaffney Jr. Software function, source lines of code, and development effort prediction: a software science validation. *Software Engineering, IEEE Transactions on*, (6):639–648, 1983.

[2] A. Begel and B. Simon. Novice software developers, all over again. In *Proceedings of the Fourth international Workshop on Computing Education Research*, pages 3–14. ACM, 2008.

[3] D. Bell, T. Hall, J. E. Hannay, D. Pfahl, and S. T. Acuna. Software engineering group work: personality, patterns and performance. In *Proceedings of the 2010 Special Interest Group on Management Information System's 48th Annual Conference on Computer Personnel Research*, pages 43–47. ACM, 2010.

[4] B. Boehm and D. Port. Educating software engineering students to manage risk. In *Software Engineering, 2002. ICSE 2001. Proceedings of the 23rd International Conference on*, pages 591–600. IEEE Computer Society, 2001.

[5] M. Borrego, J. Karlin, L. D. McNair, and K. Beddoes. Team effectiveness theory from industrial and organizational psychology applied to engineering student project teams: A research review. *Journal of Engineering Education*, 102(4):472–512, 2013.

[6] C.-Y. Chen and P. P. Chong. Software engineering education: A study on conducting collaborative senior project development. *Journal of Systems and Software*, 84(3):479–491, 2011.

[7] K. Fertalj, B. Milašinović, and I. N. Kosović. Problems and experiences with student projects based on real-world problems: A case study. *Technics Technologies Education Management-TTEM*, 8(1):176–186, 2013.

[8] M. Gehrke, H. Giese, U. A. Nickel, J. Niere, M. Tichy, J. P. Wadsack, and A. Zündorf. Reporting about industrial strength software engineering courses for undergraduates. In *International Conference on Software Engineering*, pages 395–405. IEEE, 2002.

[9] É. Germain and P. N. Robillard. Towards software process patterns: An empirical analysis of the behavior of student teams. *Information and Software Technology*, 50(11):1088–1097, 2008.

[10] E. M. Goldratt. *Critical chain*. North River Press Great Barrington, MA, 1997.

[11] J. H. Hayes, T. C. Lethbridge, and D. Port. Evaluating individual contribution toward group software engineering projects. In *Software Engineering, 2003. ICSE 2003. Proceedings of the 25th International Conference on*, pages 622–627. IEEE Computer Society, 2003.

[12] O. Hazzan. Computer science students' conception of the relationship between reward (grade) and cooperation. *ACM SIGCSE Bulletin*, 35(3):178–182, 2003.

[13] I. Jacobson, P.-W. Ng, P. E. McMahon, I. Spence, and S. Lidman. *The Essence of Software Engineering: Applying the SEMAT Kernel*. Addison-Wesley, 2013.

[14] L. Johns-Boast and S. Flint. Simulating industry: An innovative software engineering capstone design course. In *Frontiers in Education Conference, 2013 IEEE*, pages 1782–1788. IEEE, 2013.

[15] M. A. Marks, J. E. Mathieu, and S. J. Zaccaro. A temporally based framework and taxonomy of team processes. *Academy of management review*, 26(3):356–376, 2001.

[16] J. T. F. on Computing Curricula Association for Computing Machinery/IEEE Computer Society. Curricula recommendations. http://www.acm.org/education/curricula-recommendations, April 2016.

[17] A. Radermacher, G. Walia, and D. Knudson. Investigating the skill gap between graduating students and industry expectations. In *Software Engineering, 2014. ICSE 2014. Proceedings of the 36th International Conference on*, pages 291–300. ACM, 2014.

[18] D. F. Rico and H. H. Sayani. Use of agile methods in software engineering education. In *Agile Conference, 2009. AGILE'09.*, pages 174–179. IEEE, 2009.

[19] J. C. Schlimmer, J. B. Fletcher, and L. A. Hermens. Team-oriented software practicum. *Education, IEEE Transactions on*, 37(2):212–220, 1994.

[20] D. A. Trytten. A design for team peer code review. *SIGCSE Bull.*, 37(1):455–459, Feb. 2005.

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

[22] L. Van Der Duim, J. Andersson, and M. Sinnema. Good practices for educational software engineering projects. In *Software Engineering, 2007. ICSE 2007. Proceedings of the 29th International Conference on*, pages 698–707. IEEE Computer Society, 2007.

[23] L. Yu. *Overcoming Challenges in Software Engineering Education: Delivering Non-Technical Knowledge and Skills*. IGI Global, 2014.