CLEI ELECTRONIC JOURNAL, VOLUME 15, NUMBER 1, PAPER 1, APRIL 2012

Uncovering Steady Advances for an Extreme Programming Course

Viviane A. Santos, Alfredo Goldman
University of São Paulo, Institute of Mathematics and Statistics,
São Paulo, Brazil, 05315-970
{vsantos, gold}@ime.usp.br

and

Carlos D. Santos
University of Brasília, Management Department,
Brasília, Brazil, 70910-900
carlosdenner@unb.br

Abstract

This paper presents an empirical study about identifying improvement actions for an eXtreme Programming course in the academic environment. This exploratory study is undertaken in two preliminary phases. These phases are part of a wider research project to develop a theory about how to continuously improve courses of similar structure and content. The first phase consists of diagnosing improvement actions from the 2010 edition of the course through a qualitative analysis of data obtained using various methods: (1) students' responses to a questionnaire with open questions; and (2) students' opinions expressed in a final agile retrospective with all members of the course. The second phase consists of an early application of the identified improvements in the 2011 edition of the course to gather lessons learned, and develop a definite case study design to be used continuously in the next courses offered. Amongst the results, we found that the use of initiatives to promote interactions between groups like Coding Dojo and Brainwritting helps students to effectively learn and share knowledge and experiences, a problem still unsolved when thinking of scaling agile methods. Also, this paper allows keeping track on what is occurring in the course.

Keywords: agile software development, extreme programming, software engineering education, improvement actions, axial coding, qualitative analysis, case study, new forms of interaction.

1 Introduction

Agile methods, which follow the values and principles of the Agile Manifesto [1], have encouraged changes in the way software is developed. These lightweight methods emphasize feedback and change through short iterations to deliver working software. They strongly consider human aspects of collaboration, teamwork, and interaction through face-to-face communication between development teams and customers [2].

Agile software development has received a great deal of attention by practitioners and academics aiming at understanding and improving effective forms to develop and deliver software. After a decade of worldwide adoption, agile methods have impacted the software industry deeply by addressing the business pressure of fast response to change and delivery, all with a people-oriented approach [2]. Likewise, software engineering education has responded to this influence through adjustments in curricula to conform to this new paradigm [3].

Many authors have reported the importance of teaching agile methods in academic environments and proposed ways to accomplish this, but most of them consider it a challenging endeavor [4] [5] [6]. The course has to incorporate a mixture of theory and practice, especially for employing agile values, principles, and practices [7]. Also, a place for building the agile environment has to be provided for students [8]. And, the presence of diverse roles, such as customer, coach, developer and mentor, and the time devoted to development work in real projects contribute to a learning environment similar to the marketplace [5] [9] [10].

Given these challenges, only a few universities succeed in offering the ideal context to teach agile methods, since hardly ever all the aspects of a professional software development environment can be mimicked in the academia [5]. For instance, some courses provide theory and practice, but do not provide space for building informative workspace. Other courses work on predefined challenges instead of real customer requirements. And in others, the course organizers accumulate the roles of the coach and the customer.

One of agile methods gaining more relevance is eXtreme Programming (XP) [2]. The first edition of XP focuses on best practices for software development through twelve practices such as the planning game, small releases, metaphor, simple design, testing, refactoring, pair programming, collective ownership, continuous integration, 40-h week, on-site customers, and coding standards [11]. In its second edition, the practices were divided in two sets: primary and corollary practices [12]. The fundamental ideas remained the same, but more human aspects were included, like energized work, sit together and whole team[1].

Since 2001, an XP course at the University of São Paulo (USP) teaches XP in a context that encompasses most aspects needed to offer a real experience of agile software development to undergraduate and graduate students. There, we undertook two exploratory studies aiming at improving it continuously.

1.1 Research Motivation and Methods

This study is part of a wider research project that aims to answer the question: "How to continuously improve the XP Laboratory course?". Before reaching the main objective, we employed two exploratory studies to integrate our understandings on what to improve and how to apply the improvement actions in the next editions of the course through students' feedback. The strategy adopted to achieve this objective consisted of two phases:

- Phase one aimed at diagnosing what to improve in the course through a qualitative data analysis of two distinct approaches: (1) intensive coding techniques on students' responses to a questionnaire with open questions[2], and (2) an agile retrospective at the end of the 2010 edition of the course with the participation of all members of the course;
- Phase two engaged an early application of one improvement action discovered at Phase one – forms of interaction – to improve students' learning in the XP Laboratory course, in a pilot case study in the 2011 edition of the course. We aimed at reporting on lessons learned from the study for a definite case study that will be further employed in the course to develop a theory on the main research problem.

These phases allowed us to establish more inclusive meanings and to refine our next sample. In the acknowledged Eisenhardt's work [13], she discusses the confusion that surrounds the distinctions on qualitative data, inductive logic, and case study research. By claiming for clarity on developing theory from cases, she combined previous work on qualitative methods, case study research, and grounded theory building to provide a highly iterative and tightly linked to data process of inducting theory from cases. Our method for answering the main research question is in line with her process, since we intend to employ an iterative process of joint data collection and analysis within a case study to further build a theory in the 2012 edition of the course.

This article presents two empirical studies, and is organized as follows. Section 2 describes the XP course at USP. Section 3 presents the diagnosing phase that combined two approaches and was conducted with the students: intensive coding techniques through responses to a questionnaire, and the final agile retrospective for establishing improvement actions. We applied coding techniques from grounded theory in the questionnaires to systematize the connections between categories and the creation of more inclusive ones. Section 4 reports the pilot case study and the lessons learnt. Section 5 concludes and discusses further work.

2 XP Laboratory: The Course Context

The discipline called XP Laboratory [5] has offered to undergraduate and graduate students of Computer Science the opportunity to learn agile software development methods in real projects. The course is composed of an instructor, meta-coaches, and students that conduct real projects with real customers. Meta-coaches are experts in agile methods that, along with the instructor, provide agile mentoring to all teams. Experienced students on software development or students that already know agile methods assume the role of coach and others work as developers. The course requires 8 hours per week of dedication by students, and provides lunch every Friday to stimulate them to save this time for the project and to be engaged in its activities.

The course provides the necessary conditions, physical and technological infrastructure, a learning environment for the students to get in touch with XP in a context similar to professional context. This means two laboratories as

[1] Team composed by members with diversified roles, like managers, leaders, developers, testers, users, etc.

[2] The open and indirect form of the questionnaire is designed to encourage students to project their views, difficulties, motivations, beliefs, attitudes or feelings about the underlying problems in the study.

working areas, material for making the collaborative workspaces [14] and estimating with planning poker [15], two rooms for planning, technical and customer approval meetings and team retrospectives [16], and lunch every Friday. Many students, engaged in agile software development research, have undertaken relevant XP studies and work in this course [14] [17] [18] [19] [20].

In the first class, the instructor presents the course objectives, structure, schedule, evaluation criteria and available projects. Each student elects the three most desired projects to assemble the teams. In addition, the coaches are chosen. In the second and third classes, XP is explained to the students. Separately, meta-coaches teach coaches how to lead agile projects. Then, students are presented to their respective work environments, projects, customers, and coaches.

3 Phase One

In the 2010 edition of the course, there were 3 experienced meta-coaches and 51 students, acting as 9 coaches and 42 developers in a total of 8 projects. One of the coaches also worked as a meta-tracker, due to his masters' research on agile tracking. He provided support to all teams in conducting appropriate tracking on their projects features. Also, there was a project with two coaches.

3.1 First Approach: Open questionnaire

The first approach corresponds to a qualitative research based on empirical data from a questionnaire given to students before the end of the course in order for them to express their faithful views of the course. The questionnaire aims at understanding, in the perspective of the students, what needs to be improved in the course.

The methodology used in this approach corresponds to the data collection by questionnaire and qualitative data analysis with intensive coding techniques from Grounded Theory (GT) [21]. Data collection was conducted in an open, non-mandatory, and anonymous form. Team members were asked to answer the following questions: What did you learn? What went right? What went wrong? What could be improved? And deliver their responses to their respective coaches. There were teams in which each member answered the questionnaire separately, but there were also teams that completed the questionnaire together and others that did not have feedback from some members. Overall, 44 questionnaires were filled out.

The way we collected the data can result in selection bias, due to the possibility of students' omission by feeling intimidated to criticize the course. To reduce the effects of selection bias, the instructor encouraged students to provide genuine feedback as a way to improve the course. We are aware that even though we tried to limit the amount selection bias, we cannot eliminate it.

For the qualitative data analysis, the coding techniques from GT seemed more suitable, since they are grounded on data, systematize the connections between categories, deal with human behavior phenomena and allow the extraction of significant aspects of the case in study.

3.1.1 Grounded Theory Qualitative Research Method

The grounded theory qualitative research method was created by Glaser and Strauss with the aim of building inductive theory through iterative and systematic (theoretical saturation) collection and analysis of empirical data [22]. Unlike other methods that use data to evaluate predefined hypotheses, this method is based on collected data to generate a theory based on them.

According to Urquhart et al. [23], this method has four key features: (1) its main purpose is theory building, (2) researcher prior knowledge should not take him/her to pre-formulated hypotheses and should not hinder observations based solely on data; (3) analysis and conceptualization are entangled through the core process of joint data collection and constant comparison; and (4) the codes are selected by a process of theoretical sampling.

We developed Fig. 1 to briefly depict the whole process of this research method. It consists of examining each provided document more than once and independently by more than one researcher to reduce bias, seeking to identify units of meaning through a process of open coding, which seeks to establish **conceptual categories**.

Next, connections among conceptual categories are made through axial coding. At this stage, it is necessary to define a logical association between categories. Thus, the constant comparison method is employed, working "back and forth" between the established categories and the original data in order to identify related phenomena, causal conditions, contexts, action/interaction strategies, and consequences of actions to develop **notional categories**. Charts can be used to depict these relationships. At this stage, it is also possible to identify one or more interesting phenomena (dimensions) within the theoretical sampling presented in various contexts to refine the next sample.

Then, after reaching theoretical saturation on several joint data collection and analysis, the selective coding is performed to identify the main categories, existing or new, to develop a descriptive narrative (theory) encompassing

them. Later, the interpretations of the theory are validated and refined through theoretical integration, which consist of relating the resulting theory to other existing theories to generate confirmations, and even formalizations.

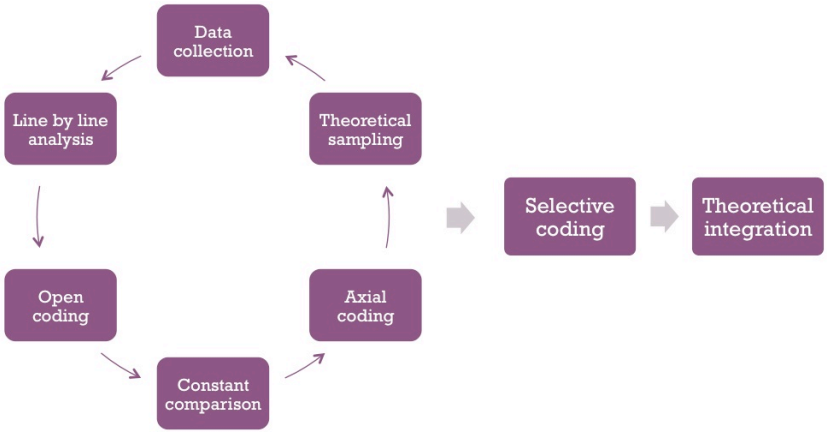

Figure 1: Grounded theory research process based on [21].

The data from the questionnaire were analyzed through open and axial coding steps. To reduce bias, each researcher performed the coding separately, and disagreements on categories were discussed to keep or discard them. As there was only one data collection in this approach, and we just applied open and axial coding steps from GT, we do not consider GT as the research method, but we are aware that the results from intensive coding helped us to refine our research and the next sample. Thus, this means our first theoretical sampling of a wider research project to discover a theory on the main theme, which is still in progress[3].

Regarding the continuity of this research, the students' point of view is very relevant to define the next joint collection and analysis of data. For this reason, we went a little further and applied the selective coding to identify the main category that could explain their point of view of the course in this edition. We emphasize that this does not mean a theory development, but theoretical sampling to decide the analytic grounds for the next sample [23].

3.1.2 Open Coding

In this step, 'in vivo codes', i.e. based on the terminology used by the informants, were used. The open coding resulted in a matrix of 33 conceptual categories, with examples of units of meaning of the original data and number of occurrences (#). Due to space limitations, Table 1 shows only some of the conceptual categories drawn from this phase[4]. The underlined text is an initial relationship between conceptual categories and it is explained in the next section. As GT establishes the need for joint collection and analysis, and in this study there was only one data collection, it was not possible to validate the conceptual categories. We intend to pursue this venue in a future study.

Table 1: Examples of conceptual categories from open coding

Id	Conceptual Category	Example of Unit of Meaning	#
1	Pair Programming	"Working in pairs facilitates the implementation of best practices and avoids inclusion of bugs. In addition, it allows for more than one person to know all the pieces of code"	24
2	Performance	"We learn to develop with agility (in terms of practice and quality)"	10
3	Code Quality	"The goal of 100% of test coverage and familiarization with new technologies (Rails) were things I found that worked well, also we were able to develop new features with better code"	6
4	Teamwork	"I learned to work collaboratively with the team"	53
5	New Technologies	"Learning many new technologies and tools"	63

[3] We intend to continue working on refining the theory in the 2012 edition of the course.

[4] To view open coding details and the list of conceptual categories, please visit http://www.ime.usp.br/~vsantos/pesquisa1/ (in Portuguese).

3.1.3 Axial Coding

In this stage, relationships among the categories, provided by students, were highlighted (Table 1). Then, we used a chart to help in visualizing these relationships, and defining the logical connections to generate notional categories. We first identified categories making a complete subgraph (cliques). Then, we identified cycles, paths that start and end at the same vertex. Lastly, we made simple bindings. Due to space constraints, we do not illustrate the resulting graph[5]. Otherwise Fig. 2 shows cliques, cycles and simple connections identified in the notional category "Planning Game".

In some categories we established simple connections with different categories, such as "Course Working Hours" with "Sharing Knowledge", "New Technologies" and "Theory on XP". However, we chose to group categories with greater relevance in the context, always seeking to be grounded in the data. Several cases of relationship did not result in grouping due to the weak connection found in the narrative of certain categories.

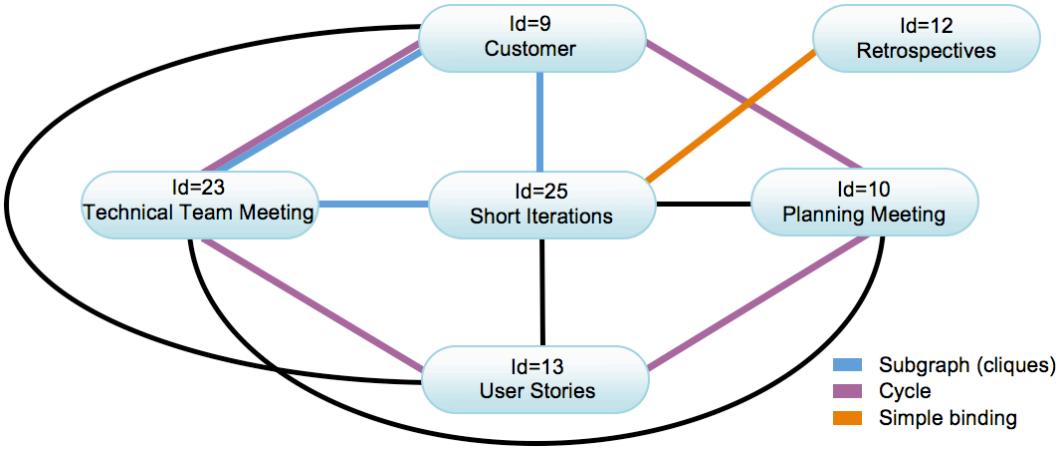

Figure 2: Logical connections among the conceptual categories to generate the "Planning Game" notional category.

After connecting the conceptual categories, notional categories were established. Table 2 shows the category identifier, notional category name, the related conceptual categories, along with their respective identifiers in the left, followed by name and number of occurrences in parentheses. To facilitate the understanding of each notional category generated by relationships among conceptual categories, the narrative around them is described as follows.

In the **Planning Game**, "Customer" involvement promotes the effectiveness of "Planning Meeting", which facilitates the understanding of the "Stories", and the design of the solution in the "Technical Team Meetings" that seek to deliver incremental value in "Short Iterations".

In the **Refactoring and Testing** category, "Tests" influence the maintainability of the code and provide the courage to perform "Refactoring". In the category **Technological Factors**, the risk of adopting "New Technologies" in the project influences team "Performance", "Code Quality" and the application of "Design Patterns", which can be mitigated with the use of "Dojo and Superparing".

Table 2: Examples of conceptual categories from open coding

Id	Notional Category	Related Conceptual Category
1	Planning Game	9) Customer (30)
		10) Planning Meeting (1)
		12) Retrospectives (11)
		13) User Stories (25)
		23) Technical Team Meeting (8)
		25) Short Iterations (15)
2	Programming Best Practices	31) Programming Best Practices (5)
3	Refactoring and Tests	6) Tests (24)
		33) Refactoring (8)
4	Communication and Feedback	1) Pair Programming (24)
		4) Teamwork (53)
		8) Collaborative Workspace (8)

[5] The resulting graph with the existing groupings separated by color is available at http://www.ime.usp.br/~vsantos/pesquisa1/grafo-cat-conceituais.png (in Portuguese).

Id	Notional Category	Related Conceptual Category
		11) Communication (13)
		17) Tracking (27)
		18) Course Schedules (4)
		19) Lunch (9)
		22) Standup Meeting (3)
		26) Estimates (21)
		30) Commitment (5)
5	Technological Factors	2) Productivity (10)
		3) Code Quality (6)
		5) New Technologies (63)
		27) Dojo and Superparing (4)
		32) Design Patterns (2)
6	Form of the Course	14) Course Hours (29)
		15) Theory on XP (24)
		20) Evaluation (6)
		28) Teacher Involvement and Meta-Coaches (8)
7	Forms of Interaction	7) Environment (9)
		21) Coaches Meeting (5)
		24) Knowledge Sharing (8)
		29) Coach (18)
8	Software Development Environment	16) Software Development Environment (10)

In the **Communication and Feedback** category, the team's involvement in “Pair Programming”, in “Standup Meeting”, in “Estimates” definition, in “Lunch”, in update of the “Collaborative Workspace” and the charts for “Tracking” influence positively “Communication”, “Commitment”, and punctuality of members in the “Course Schedule” to the smooth running of the “Teamwork”.

The **Form of the Course** requires more “Teacher and Meta-Coaches Involvement”, “Course Working Hours”, “Theory on XP” and better “Evaluation”. **Forms of Interaction** among teams should not only occur in the “Coaches Meeting” because of the trust the team has in the “Coach”, so that a better use of the "Environment" helps to intensify "Knowledge Sharing".

The **Software Development Environment** category did not fit the above-mentioned logic. However, students reported its connection to the project progress. We concluded that an unfeasible **Software Development Environment** affects the progress of the **Planning Game**. Note that, through constant comparison, we identified a relationship between two notional categories.

Similarly, the category **Programming Best Practices** did not bind strongly to any other category to establish a group. Nevertheless, students reported that these practices facilitate the development and maintenance of source code. So, **Programming Best Practices** favors the smooth running of the **Planning Game**, influences the **Communication and Feedback**, and **Technological Factors** affects **Refactoring and Testing**.

After the definition of notional categories, according to the protocol, the constant comparison method is applied between the categories and the original data to establish the categories dimensions. Table 3 presents the resulting dimensions.

Table 3: Dimensions of notional categories

Id	Notional Category	Dimensions	
		Connectedness	**Distinctiveness**
1	Planning Game	Team gradual efficiency	Monitoring of software development deficiencies
2	Programming Best Practices	Recognition of the value of their application	Incentives for implementation
3	Refactoring and Tests	Practices appreciation	Degree of trouble in applying
4	Communication and Feedback	Team gradual efficiency	Monitoring of software development deficiencies
5	Technological Factors	Technological factors appreciation	Level of experience in market technologies
6	Form of the Course	Need for more working hours, more theory and further monitoring	Students initiative
7	Forms of Interaction	Leveling learning	Monitoring of software development

Id	Notional Category	Dimensions	
		Connectedness	Distinctiveness
8	Software Development Environment	Need to initially be ready to software development	deficiencies Level of experience in market technologies

3.1.4 Selective Coding

This coding technique has been applied to extract students' point of view about the 2010 edition of the course. This does not imply theory development, but helps deciding the analytic grounds for the next stages of joint data collection and analysis, and to identify what needs to be improved in the course.

Thus, the categories of **Planning Game**, **Programming Best Practices**, **Refactoring and Testing**, **Communication and Feedback**, **Technological Factors**, **Form of the Course**, **Forms of Interaction** and **Software Development Environment** can be interpreted as essential to the success of the course and the satisfaction of students. Every category is part of the story, though none of them captures it completely. For this reason, another abstract term or phrase is required, a conceptual idea under which all categories are included. Hence, we conclude that the resulting main category is "Means to Support the Learning in the XP Laboratory course".

Consequently, we integrate the categories and present the narrative to improve the course as follows. The XP Laboratory course is of great value to students, as it provides learning through the development of software project using agile methods in a context closer to the marketplace. As a student stated, "The course gave me much opportunity to learn how to manage a team and negotiate projects with a client". However, it needs additional support for a more effective learning of new technologies, concepts and practices with continuous student evaluation, reconsideration of the course prerequisites, and working hours. As another student declared, "The main problem of course is the accumulation of things one should learn in short time. The main cause is the lack of pre-requisite courses that prepare us to this course. In addition, as we have to learn new technologies, when we get better and faster, the semester ends. Perhaps the ideal would be to change to annual course and increase the credits to twelve. Certainly we would have better projects and more satisfied customers". Concerning students' evaluation, a student reported, "Because it is a totally different discipline amongst traditional, feedback is necessary to help us adapt to the discipline". Regarding knowledge sharing, a student summed up, "I think there should be more knowledge sharing among teams. Although the focus is on work, each team had different experiences that should have been shared with all".

3.2 Second Approach: Final retrospective

In the last class of the 2010 edition of the course, we conducted an adapted agile retrospective with all participants. The rationale for collecting data by this second approach is that students verbally expressed their opinions in the class, instead of collecting written evidence as gathered in the first approach. We aimed at triangulating data and generating reflections on the course to establish a shared vision of improvements needed for future editions. To accomplish that, we followed the steps described next.

3.2.1 Methodology

The final agile retrospective occurred in a classroom with sessions of seven minutes for a discussion of each topic described in Table 4. Teacher and meta-coaches managed and recorded the sessions in a report. Another seven-minute session could be used for a deeper discussion of the same topic to extend the discussion, if judged necessary.

This agile retrospective used an approach called Fishbowl discussion[6], due to the large number of participants and little time for discussion (two-hour class). This disciplined approach consists of five chairs arranged in front of the room so students interested in reporting anything about the topic under discussion in the session could sit down and express their opinions. The discussions could occur only between people sitting in chairs. The rest of the room should be silent, without any parallel conversation. Among the five seats, there should be always a free chair for the next student interested in discussing the topic. Hence, if someone seats in the remaining empty chair, one of the four seats would release its own chair, as he/she already laid out his/her vision.

Before starting the sessions, the students suggested 17 topics (issues) for discussion. Then, the entire class voted on themes they were most interested in discussing.

[6] http://en.wikipedia.org/wiki/Fishbowl_(conversation)

3.2.2 Results

Due to time constraints, the five most voted topics were discussed in the dynamic. Table 4 lists the issues. The number of votes are in parentheses, exemplary transcriptions of the report and the number of sessions (#). Whenever a session is finished, the considerations reported by the meta-coach were read and confirmed or updated by the participants until consensus was reached. The generated report describes the points addressed in the final retrospective.

Table 4: Topics covered in the whole-class retrospective

Id	Topic	Report Transcriptions	#
1	Theoretical part (29)	"Spread the classes would turn theory into practice."	1
2	Schedule of discipline and dedication expected off-hours (25)	"Maybe the discipline deserves more credit and more classes per week."	1
3	Learning of technologies (16)	"The necessity of many technologies that were not dominated (e.g. Eclipse, Ruby, Rails, Git, etc.)."	1
4	Participation of meta-coaches and teacher (12)	"Have more proximity between the group and meta-coaches to have more time to feel what is OK and what is not"	2
5	Evaluation criteria (10)	"Lack of feedback and the subjective criteria leaves everything unstuck. Very difficult to improve."	2

The researchers also were participant-observers in the dynamic of study. Even discussing just five topics, many issues were also mentioned. It allowed the development of a clearer notion of the most urgent course needs based on what was expressed by the students.

After analyzing the report, we can conclude that in the opinion of the students, they need more monitoring, support, and feedback from the responsible of the course in relation to learning XP, the technologies involved and their deficiencies in project development.

3.3 Discussion of the Findings

We realize that both approaches unfolded an intersection among the findings, which means enhance learning of concepts, practices and technologies; increase the number of credits; improve interaction among members of the course; and establish objective evaluation criteria. Even with the difficulty to distill all the data and analysis materials into a journal paper, we preserved the chain of evidence by providing the extraction of interesting expressions from the questionnaires and the report transcriptions during data analysis.

The combination of approaches provides triangulation of data for better results. During the observation of participants, the projects of the course presented varying complexities as well as teams that managed to overcome the problems completely, whereas others had been able to partially overcome challenges, and others that could not overcome problems whatsoever.

Thus, the actions for improvement proposed by the students are forms of interaction (dynamics) of Coding Dojo[7] [24], retrospectives[8], and mini-lectures[9] to be explored with all students for encouraging adoption, learning and sharing of values, principles and practices of XP; to provide a more detailed exposition of the subject and detection/treatment of students' deficiencies during the agile software development; and to facilitate the preparation of the software development environment.

Other improvement actions consist of increasing the number of credits, and hence the course working hours. Since the success of an agile project depends on the type of project and team, other improvement actions consist of reconsidering the way of making teams [25][10] and the criteria of the periodic evaluations, as students must be continually evaluated during the course to provide the necessary support to the difficulties encountered.

This study represents the start of an in-depth research about continuous improvement in agile methods and education. As there was no theoretical saturation, since we did only one data collection and, at the end of the course,

[7] Coding Dojos are mostly used to level knowledge and foster insights among participants. Students suggested this form of interaction in both approaches, which turned into a conceptual category (id=27) and inside a topic of the whole-class retrospective (id=3).

[8] Retrospectives are mostly used to raise feedback and was suggested by the students in both approaches in conceptual category (id=12) and whole-class retrospective topic (id=3).

[9] Mini-lectures were raised in conceptual category (id=15) and whole-class retrospective topic (id=1).

[10] The authors state that the most effective agile teams are those that consider the diversity of its members, confirming the XP whole team practice [12].

the researchers intend to use these preliminary results to better specify the next joint collection (improving the questionnaires, theoretical sampling, using other types of collection, etc.) and analysis throughout the course to confirm or refute the results. The authors also consider building a repository of experience as described in [26] and apply organizational learning techniques [27] to identify effective ways to generate continuous improvements in agile teams/organizations.

4 Phase Two

After diagnosing improvement actions, we conducted a pilot case study in the 2011 edition of the course, from March to June, to get an early application of the forms of interaction with all students (first improvement action raised by the previous section), and capture lessons learned for a definitive case study the authors plan to undertake.

According to Yin [28], pilot cases help the researchers to prepare for data collection and refine data collection plans with respect to both the content of the data and the procedures to be followed. A draft protocol should be provided to represent the topics of interest to the case study.

Pilot case study allows the researchers to check that the research propositions still make sense and that the interview questions are actually addressing the propositions. If there is any problem with the interview questions, they may need to be modified.

Regarding the means to support the learning process in XP Laboratory course, several authors state that knowledge should be apprehended for action and the support for learning in action should be provided [29] [30] [31]. Also, the need for a context or space (*ba*) for learning is outlined by Nonaka and Takeuchi [32], which does not mean just physical structure, but also conditions, stimuli, and practices for knowledge creation and transformation. These authors refer to the relevance of knowledge context, which affects the propensity to change in behavior and learn through cognitive developments.

Since agile methods are linked to action and are more people-oriented, agile software development is better absorbed by putting values and principles into practice. The improvement actions raised by the previous study (Section 3.3) are in line with the concern to support and promote the learning of XP. Then, the use of diversified practices tends to improve learning.

4.1 Draft Protocol

This section presents a draft protocol for the pilot case study, including case selection, scope of the pilot inquiry, research question, pilot case design, preliminary research propositions, field procedures, and data collection plan.

4.1.1 Case Selection

The selection criteria for the pilot case study in the 2011 edition of the XP Laboratory course are described as follows. First, the Institute of Mathematics and Statistics at USP is considered a reference in computer science. Second, the relevance of the course in our country, since there are few XP courses in the Brazilian software engineering education. Moreover, we consider the acknowledged instructor-researcher expertise on agile methods.

Also, the convenience and availability of the context for answering the research question. The potential to predict similar results (a literal replication), since it will be run in another class but in the same course, when replicating to other case studies. The great potential to learn from the research context. The congenial relationship established between course participants and researchers. Yet, the participants' awareness of the early stage of the research.

4.1.2 Scope of Pilot Inquiry

This inquiry aims at covering substantive issues that include the evaluation of an improvement action defined in Section 3.3, which is the application of several forms of interaction with all students, and also methodological issues like refining field procedures and data collection plan.

The increase in the course length and credits was not possible to be accomplished in this edition of the course because it is a bureaucratic and time-consuming process at the university. Additionally, the XP practice of making the whole team [12] was quite difficult to achieve, as most students enrolled in this edition have similar roles and few have extra experience in software development, tests, user-requirements, and so on. The ones with extra experience and the ones that have already attended the discipline in the basic form were assigned to become coaches. Yet, due to space limitations in this work, we did not present the criteria for periodic evaluation and the repository of lessons learned. We intend to engage both in a future study.

4.1.3 Research Question

Derived from the main category emerged by the selective coding (Section 3.1.4), "Means to support the learning in the XP Laboratory course" and considering the scope of the pilot inquiry in the previous section, phase two's research question is:

- How to effectively apply forms of interaction, dynamics such as Coding Dojo, retrospectives, and mini-lectures, to support learning in the XP Laboratory course?

4.1.4 Pilot Case Study Design

This pilot case is a single-case design, since we have one context, the XP Laboratory course, with seven teams composed to develop real projects with real customers and requirements.

In the 2011 edition, the course has one instructor-researcher, two meta-coaches (including the other researcher) and forty-five students, which among these, ten served as coaches. There are two members in one project and three members in another project who are acting as coaches. Because of lack of availability, two meta-coaches and the meta-tracker from the 2010 edition were not present in this edition of the course.

These seven teams were separated in two laboratories according to the technology used in the project to encourage knowledge sharing among teams due to proximity: java client-server (desktop) and web applications.

4.1.5 Preliminary research proposition

The following research proposition considers the instructor-researcher experience on teaching XP for about 10 years, and the forms of interaction (dynamics) as a means to support learning XP:

- RP1. Dynamics like Coding Dojo, retrospectives and mini-lectures with all influence positively in the support for learning XP[11] in the course.

This research proposition imply encouraging students to adopt, learn and share agile values, principles and practices, providing detailed explanations and detection/treatment of students' challenges during the agile software development, and facilitating the software development process for the teams.

4.1.6 Field Procedures

The field procedure for this pilot case is composed of:

- To explore dynamics with all to support the learning of agile methods;
- To gather partial feedbacks from students through coaches meetings, teams' stand-up meetings, teams' retrospectives, whole-class retrospective, evaluation of the course with open-ended questionnaires, and dynamics ending up with retrospectives;
- To provide partial feedbacks to students through coaches meetings, teams retrospectives, grades, and informal communications;
- To take actions to support students' issues from retrospectives, from requests at mailing lists, and from informal communication;
- To gather final feedbacks from students through interviews, and whole-class retrospective;
- To analyze the level of support achieved through the students' perceptions;
- To refine the proposed improvement actions and research design.

4.1.7 Data Collection Plan

We have collected data via observations, interviews with open-ended and semi-structured questionnaires, and documentation as described in Table 5. The observations lasted from 10 minutes to 2 hours. We took field notes and pictures from the execution of dynamics with all at the one-hour classes, coaches meetings, XP practices within teams and the collaborative workspaces.

To help the researchers in identifying important areas that may have been overlooked, we conducted interviews with open-ended questions, so respondents can add any information they consider pertinent [33].

[11] Support for learning XP means encouraging students to adopt learn and share agile values, principles and practices, providing detailed explanations and detection/treatment of students' challenges during the agile software development, and facilitating the software development process for the teams.

The questions presented in interview guides refer to team learning, course improvements, forms of interaction (dynamics), and knowledge sharing within and between teams (which refers to an ongoing study of the authors).

The chain of evidence is preserved by the extraction of interesting expressions from transcribed field notes, interviews and dialogues from dynamics with all at the one-hour classes, coaches meetings, XP practices within teams and the collaborative workspaces that report on lessons learned for both substantive and methodological issues [28].

4.1.8 Limitations of the study

A possible research limitation is the influence of researchers in the course as participant-observers, since they also assume the roles of instructor and meta-coach. We minimized this limitation by reporting our preliminary findings to the other meta-coach (not involved in the research and experienced in this role), and one experienced meta-coach from the previous edition that did not participate in this edition. We emphasize the limitation of our results as they are driven to academic environments. Another limitation is the selection bias described in Section 3.1.

Table 5: Data collection plan for the pilot case study

Source	Data Collected	When	Description
Observation	Coaches meetings	Every Wednesday at 01:40pm	Weekly meeting involving coaches, meta-coaches and instructor for project review, for sharing of issues/solutions and for requesting support.
	One-hour classes	Every Friday at 12:00pm	New forms of interaction with everyone were undertaken at classroom where lunch is provided.
	XP practices	Once a week	We observed stand-up meetings, collaborative workspaces, planning meetings, customer approval meetings and retrospectives within the teams.
Interview	Open-ended questionnaire	Once a month	Interview guide: What did the team learn? What went right? What went wrong? What could be improved? Any lesson learned from the iteration? Any knowledge sharing among other teams? Respondents: all teams in the first interview, but after that, just a few teams answered.
	Semi-structured questionnaire	After applying all the new forms of interaction	Interview guide: Which dynamic(s) did you like more? And why? Which dynamic(s) did you like less? And why? The dynamics were useful? With the dynamics, in your opinion, did the environment become more willing to communication and sharing among people from other teams? If 'Yes', how? What does the lunchtime at Fridays mean to you? (Options: Opportunity to learn something new; A moment to know other people and abilities; Opportunity to clear doubts; Waste of time; Other). What is your role in your project? (Options: Coach; Developer). Respondents: 24 (53% of the class)
Documentation	Collaborative workspaces	Twice a week	Progress charts, velocity metrics, project tracking, user story cards, etc.
	Project Wiki	Once a week	Information about the project, like project description, issues, team, architecture, tools, lessons learnt, etc.
	Mailing Lists	Every day	Share of practical knowledge in two main mailing lists and other group-specific mailing lists.

4.2 Conduct the Pilot Case

In this section, we discuss how we conducted the pilot case by following the formal field procedures, data collection plan, and report characteristic episodes, which show aspects of the improvement actions and protocol refinement.

4.2.1 Overview of the 2011 Edition of the Course

Projects available for choosing were either open source or university internal projects. The chosen projects, described in Table 6, are CHOReOS V&V, CHOReOS Middleware, Archimedes, Graduate Admission, Online Programming Exercises, Arquigrafia and Mezuro. These projects held different contexts and teams.

CHOReOS project is an international consortium open source project, which aims at addressing the challenges inherent of the Ultra-Large-Scale (ULS) Future Internet of software services. V&V and Middleware are subprojects developed at the Institute of Mathematics and Statistics, University of São Paulo by M. Sc. and PhD students. Due to the relevance, innovation and uncertainty of both subprojects, more than one coach composed their teams.

The environment for the course is separated in two laboratories. Eclipse Laboratory is the space for java client-server projects, such as CHOReOS V&V, CHOReOS Middleware and Archimedes. CEC Laboratory is the space for web-based projects, such as Graduate Admission, Online Programming Exercises, Arquigrafia and Mezuro. In both laboratories, it is provided a permanent working area for each project during the course and all material support for the collaborative workspace is offered. In short, students prefer to have their own development area.

The research aim was presented to the students and the instructor emphasized the need to gather genuine feedback from them to improve the course.

Table 6: Overview of the projects developed during the 2011 edition of XP Laboratory

Project	Description	Team Characteristics				
		Size	No. of coaches	Experience on XP	Experience on technologies	Experience on customer requirements
CHOReOS V&V[12]	This project aims at developing a framework for automated testing of choreographies.	4	2	Medium	Medium	Low
CHOReOS Middleware[9]	This project aims at implementing service middleware support to enabling the deployment of adaptable, QoS-aware choreographies in the ULS Future Internet.	7	3	Medium to Advanced	Medium to Low	Very Low
Archimedes[13]	This legacy project is a free and open source CAD (Computer Aided Design) software.	6	1	Low	Medium	Medium to Low
Mezuro[14]	Open source project that aims to allow users to submit and evaluate their software source code.	5	1	Medium to Low	Low	Low
Arquigrafia[15]	Open source collaborative environment for sharing architecture pictures that is built under Groupware Workbench, a set of legacy components aimed at social interaction and collective intelligence for building collaborative applications on Web 2.0.	4	1	Low	Low	Medium
Graduate Admission	University internal project aimed at creating an electronic registration system for M.Sc. and PhD students.	8	1	Low	Medium	Medium
Online Programming Exercises	University legacy system is focused on testing students programming exercises for several disciplines at the Computer Science courses.	8	1	Low	Medium	Medium

4.2.2 Form of the course

XP practices within teams, such as standup meetings, planning meetings, customer meetings, team retrospectives (Fig. 3) and tracking tasks help us identify teams' context, challenges, issues, adherence to agile values and principles, and so on. For instance, Mezuro's team was in trouble to get Ruby on Rails (RoR) started and it was raised as a negative point in the team retrospective. While they were figuring out how to solve their deficiency in the programming language, we suggested them a Coding Dojo with another experienced member from other team on RoR. After negotiating with the expert, they scheduled the dynamic out of class time and actually made a Superparing[16] instead of Coding Dojo, so that they could learn RoR and start writing source code.

We also engaged in coaches' meetings. In these meetings, the instructor advises coaches on dealing with problems perceived in their projects and coaches provide solutions for problems presented by others, besides the

[12] http://www.choreos.eu/bin/Discover/TheCHOReOSsolution
[13] http://www.archimedes.org.br/
[14] http://softwarelivre.org/mezuro
[15] http://www.arquigrafia.org.br
[16] *Superpairing* is a Coding Dojo considering the development of actual project stories.

instructor and meta-coaches participation. For instance, a coach from CHOReOS Middleware provided to Mezuro's team a metric for tracking estimating errors, known as *PokeBalls*[17], depicted in Fig. 4 at the side of the story tasks.

Figure 3: A team retrospective.

Figure 4: Mezuro's collaborative workspace.

4.2.3 New Forms of Interaction

As raised by the previous improvement actions (Sections 3.1 and 3.2), the above practices are not sufficient to stimulate and support learning of XP, so other stimuli (Section 3.3) were included in this edition. For this reason, we defined the lunchtime provided for the entire class on Fridays as one-hour class, so we were able to present mini-lectures on specific topics and apply dynamics with all for improving learning and interaction.

Next, we describe the findings for the suggested dynamics of mini-lectures, Coding Dojo and whole-class retrospectives (Fishbowl discussion). Moreover, we went a little further and applied other meaningful dynamics commonly used in agile methods conferences, known as Brainwritting, Lightning talks, Birds of a feather session and Starfish diagram. In the pilot case, we could analyze how to adapt these forms of interaction to the course.

Mini-lectures. Short presentations with the purpose of leveling knowledge about XP at the class. After the initial overview of XP, lunchtime at Fridays was accompanied by mini-lectures. The first one was about agile planning. At the end of each mini-lecture, instructor and meta-coaches along with students suggested topics of interest for the next mini-lecture and then voted. The most voted topic was chosen for presentation on the next week.

This initiative was praised by most students, but an interesting perception we realized was the trouble in students suggesting and also choosing a topic, especially the beginners. Regarding the trouble in voting topics, some students said, "(…) It would be better to leave the vote open during the week, so we could search and analyze the topics before voting", "Yes, I agree, we could be more aware of the topics first", "I would suggest a change in the voting schema by detailing each topic in the wiki and letting it available a week before voting".

Then, the next mini-lectures were about automated tests for web, planning and estimation with planning poker, project tracking, Test-Driven Development (TDD) [34] and creativity in software development. Many other interesting topics, like "How to write stories", "Communication with the customer", "Beauty of source code", "Refactoring", "Continuous integration" were not selected due to lack of votes.

The following statements reflect the need for more preparation and articulation among topics: "The first mini-lecture about tests was a little confusing and poorly prepared"; "In my opinion, mini-lectures should focus also on 'how to do' and not just 'what to do'"; and "It should be about everything we need to learn by doing."

Coding Dojo[18]. This is a meeting to train code programming and to improve skills [24]. We proposed the *Randori* type of Coding Dojo, where a challenge is selected, and then a list of participants of the audience is defined. It starts by a coding pair (driver and copilot) trying to solve the challenge within a timebox of 5 minutes, using TDD and BabySteps[19]. While the tests are failing, the audience may not interact. After succeeding, the audience may help the

[17] Pokeballs is a metric to identify which tasks occur estimation errors. On the side of each ongoing task, a blue circle is drawn for each estimated time. In the update, the amount worked done is filled with a dot inside each circle. If more hours are needed, red circles are added and filled in the same way. When the task is finished, a label is made.

[18] http://codingdojo.org

[19] BabySteps mean the next step should always be as small as possible.

coding pair in refactoring the code. At the end of the timebox, the driver goes back to the audience, the copilot becomes the driver and the next one of the list steps up to be copilot.

Two experienced coaches of the course conducted our first Coding Dojo. They explained the dynamic and proposed three challenges to solve in RoR for the audience to choose. It took almost 20 minutes of a one-hour class. Then, when the challenge was chosen, they started the Coding Dojo to show how it works. The chosen challenge was a greedy strategy for getting the most valuable gem with least weight.

After 60 minutes, we finished the Coding Dojo and made a retrospective to know how to improve it for the next time and get some feedback. Most students considered it very interesting and worthy, especially the beginners, as one said in a later interview, "Dojo is a great way to learn new programming languages". Others that already knew it provided relevant critics. Many students wrote in pink post-its complaints about the short time and the great audience, in their viewpoint, the dynamic should be adapted to a class of about 40 people. Also, 5 minutes for iteration were considered short, which caused much change and low productivity. And some stated that it took too long to effectively start.

With little interaction, the audience was also raised as a negative aspect. Some students stated that it might be related to other negative aspects raised like unfamiliarity with RoR, a minority number of participants due to short time and great audience, trouble in seeing the code in the projector. As one wrote, "Syntax highlight was bad (some elements were too bright) and we could not see it".

Regarding the chosen challenge, there were some different opinions, some liked the selected challenge and found it appropriate for the dynamic, as a student said: "The challenge was simple, but interesting". And others said that it was not suitable to solve it in 40 minutes, one said: "The proposed challenge was much complex to this Dojo". Even with the points to improve, many students stressed that it was much valuable to know how this dynamic works and learn by doing (or seeing other doing) and RoR, even though an overview.

Then, we observed some teams undertaking other types of Coding Dojos, like Kata[20] and *Superpairing*, with their members to improve learning and solving project issues together. The Arquigrafia team also used Kata Coding Dojos with the customer (a technical customer) to better understand the project legacy code, see Fig. 5. This team also found worthy to make team *Superpairing* some times to level the programming and code knowledge.

Brainwritting[21]. This dynamic caused a great impact on students, since all were meant to interact. Before this, the lunchtime had been considered a momentum for learning in a passive way. We encouraged cross-team interaction by making seven circles of chairs (one for each project), naming each circle with each project name, and separating five story cards, pens and adhesives of red and green dots for each circle.

Figure 5: Arquigrafia's Kata Coding Dojo with technical customer (the one that is presenting the code).

When the students arrived for the class, we conducted them to their project circle. While they were having lunch, we explained the dynamic. The group then discussed and elected five project issues in each story card in 10 minutes. After that, team members, except (one) coach, stood up and spread up to other circles, without concentrating much people from the same team on a single circle. The coach stayed in the circle to explain the project issues and clear up doubts.

[20] Kata Coding Dojo means a presenter solving a programming challenge by using TDD and Baby Steps.

[21] Presented in a tutorial at the 12th International Conference on Agile Software Development (XP 2011), entitled "Retrospectives in action" and conducted by Patrick Kua and Nick Oostvogels (http://xp2011.org/program?sid=416&o=1)

The five cards are shared with the newcomers of the circle, with 2 minutes to read the card and write a proposed solution for the issue. As time passes, each one hands the card to the colleague in the left and then gets the other card from the colleague of the right. They keep on doing that until they end up with the first card they received.

Then, they return to their project circle, the coach reads the content of each card and the team discusses the proposed solutions. They use the adhesives to elect good solutions with green dots and mark inappropriate solutions with red dots. Fig. 6 shows the teams circles discussing their issues and two cards with green and red dots. Finally, we made a retrospective to reflect on and improve the dynamic.

Many positive aspects were highlighted, like: "Insight for good ideas", "Reflect on project issues", "Some suggestions were very useful", "Opportunity to help other groups", and "Exchange experience among teams".

Some negative aspects raised were "Longstanding. We have work to do!", "Short time for discussion", "It doesn't work for very technical/specific issues", "You end up reading others' opinions before writing yours", "Short time for thinking in solutions", "The lack of project context compromises the efficacy of the proposed ideas", "Shallow knowledge of project issues", "Proposed solutions already considered by the team".

After analyzing the semi-structured questionnaire, which occurred after all dynamics, the students provided some relevant statements, "Brainwritting was the only dynamic in which forced us to interact with other teams", "(…) other dynamics were applied to generic themes aiming at demonstrating, but Brainwritting was focused on project issues itself", "I have lost Brainwritting that most impressed my team. After, in the stand-up meeting, the team incorporated some suggestions to solve project issues", "It is hard to suggest a solution without knowing the context", "(…) exchange [of knowledge] would be better if there were more turnover among groups", "Our tracking was influenced by Brainwritting. Now, we are tracking small progress to enhance the moral of the team".

Fig. 7 depicts the Online Programming Exercises team discussing, in their stand-up meeting, the proposed solutions for their issues raised in the Brainwritting.

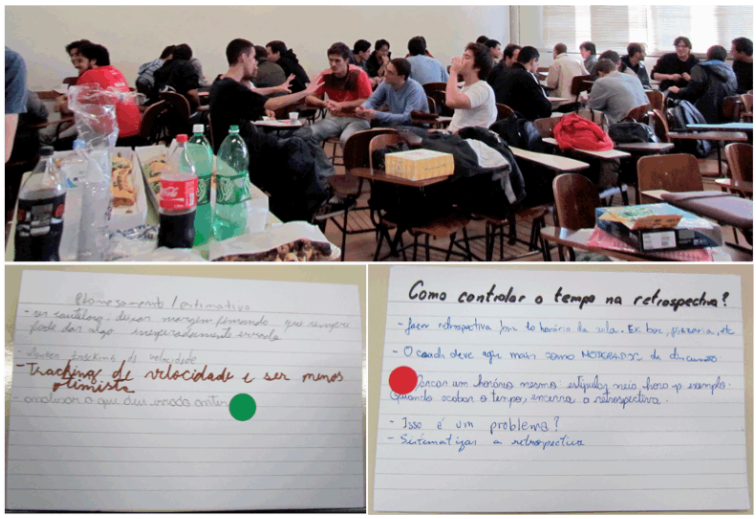

Figure 6: Brainwritting at top and two project issue cards with dots.

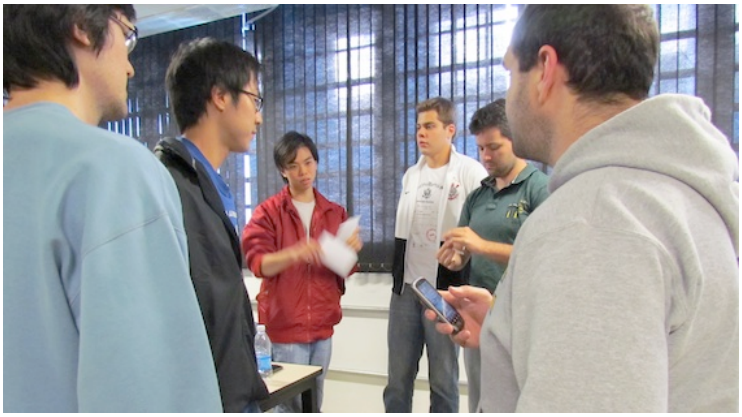

Figure 7: Solutions proposed at Brainwritting being cited at stand-up meeting.

Lightning Talks. Short presentations usually given at agile methods conferences. For the course, each group should elect a representative to present an experience of the project (any subject) that worked well in 5 minutes. The idea is to share good experiences with everyone. Next, if appropriate, a brief discussion of the experiences may occur.

Each project representative presented one (or more than one) good experience. Some used slides and others just talked to the class. When they finished, some doubts were cleared up.

At the end, we invited the students to give feedback on the presentations in the board by marking with (x) in a retrospective timeline [15]. They filled in how they felt about each presentation, good (+) or bad (-), to get a general feedback for the overall sentiment of the class. After that, the instructor drew a trend through them to establish a real idea of how the presentations went. Overall, they were positive, but a few were considered bad.

Some students criticized the use of retrospective timeline to get perception. They stressed that knowledge is always incremented, so the negative signal is not appropriate. A student said "I disagree in marking at (+) and (-), because there is no negative presentation". Others outlined that people may feel intimidated to mark a negative feedback, saying that "People won't feel comfortable in assigning which ones were bad". At the end, we identified much consensus in their opinion, this could be related to the possibility of bias, as students can read others opinion before giving their feedback. As student said: "The more opinions get to the end, the more they get similar".

Whole-class retrospectives. We made two retrospectives in the form of Fishbowl discussions (Fig. 8) with the entire class to gather feedback from the students about the course. This type of retrospective allows for disciplined and quick discussion in environments with great audience. As one student outlined: "It stimulates interaction between groups, without making a mess".

The first whole-class retrospective occurred after mini-lectures and the Coding Dojo. Students proposed free themes for discussion and voted. The fifth most voted themes were: "How to be agile and not do agile?", "Creativity in Software Development", "Refactoring", "Tracking" and "Estimation".

Many students stated that they liked the dynamic, as students stated: "It is a simple way to cause a chain reaction after the first participant takes part to talk about a subject"; and "(…) anyone who had knowledge about a certain topic could contribute".

However others said that the interaction depends much more on students initiatives to discuss the subjects, one said "I think there was little participation in Fishbowl, the discussed ideas were restricted just to more participative people", other stated also, "(…) some interact in front, and others just keep on watching, I think it didn't favored so much". Concerning the lack of interaction, some said that some topics did not attract their attention or contribute to their projects, for instance "The dynamics of fishbowl failed to add new actions, but reiterated the actions that were already working".

The second whole-class retrospective was at the end of the course and we focused on what to improve, as students had a broad view of it. Three weeks before the last retrospective with all, we put up in each laboratory a paper with a Starfish diagram, illustrated in Fig. 9, to get feedback of what to stop or start doing, more or less of. The students had previous time to think about the course and how to improve it. These feedbacks were used in the final retrospective as we categorized and discussed all of them, as well as specified actions for improvement.

The resulting categories for the Fishbowl discussion, which are exhibited in Fig. 8, were (1) Food, (2) Dynamics, (3) Workshops, (4) Interaction among groups, (5) Mini-lectures, (6) CEC Laboratory, (7) Students evaluation and (8) Legacy code. Since in this work we are focused on the forms of interaction, we analyzed categories (2), (3), (4) and (5).

Regarding the dynamics, students report that they should continue, but they should be improved to work well in a one-hour class, since the time in this discipline is too scarce. Dynamics should be focused and coherent to most audience. Many of them appreciated Starfish diagram and said "There should be more Starfish since the beginning". They suggested the inclusion of Birds of a Feather[22] sessions to group people with shared interests.

Two students had workshops approved for presentation in the conference Agile Brazil 2011[23] and they suggested on their own to present the workshops for the class as a way to try them out. So, we had a workshop of TDD and another of two games stimulating the employment of agile values and principles. In the perception of students, both experiences were considered very valuable for their learning, and they suggested considering longer activities like these, out of the development time, to improve learning.

[22] Birds of a feather (BoF) is an informal discussion group usually held on agile conferences, where the attendees group together based on shared interests, http://en.wikipedia.org/wiki/Birds_of_a_Feather_(computing).

[23] http://www.agilebrazil.com

Figure 8: A Fishbowl discussion in a whole-class retrospective.

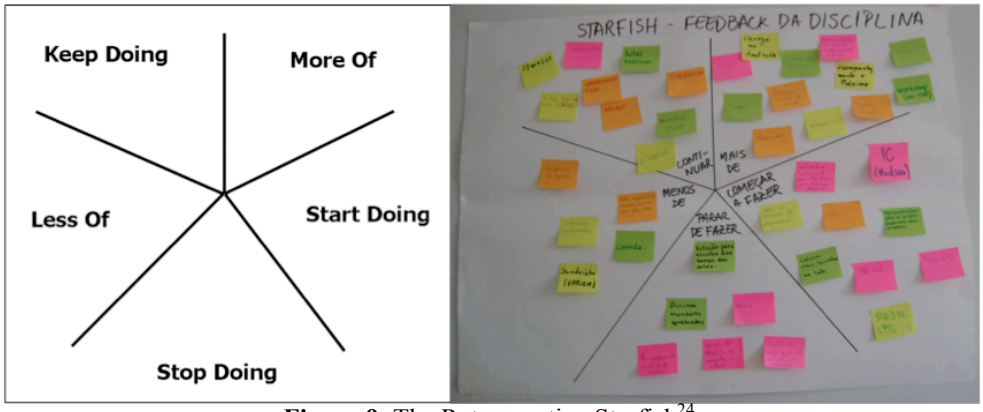

Figure 9: The Retrospective Starfish[24].

About the interaction among groups, they also suggested to know more about other projects, such as presentations. A student stated, "It would be interesting to engage collaboration among projects with similar problems". They stressed as well that mini-lectures should be kept, but satisfying indispensable topics that should come first and then establishing free topics one week before voting. They stated that in the beginning of the course they would not be able to suggest or even choose the right topic for them. The overview of XP is not sufficient for students to decide what they need to scrutinize in the mini-lectures. One of them said, "I think mini-lectures should be organized by mandatory themes, like planning, estimating, tracking, TDD, refactoring and writing stories. Then we could provide other themes, since after having classes and working on the project help us to get 'mature' on XP". Another student recommended "intercalating dynamics with mini-lectures".

4.3 Research Design Refinement from the Pilot Case

This pilot case study acts as a prelude to a new case study. It was relevant to amend improvement actions taken and field procedures. As stated by most students, using various forms of interaction help them to learn and share knowledge and experiences, because each practice reaches specific purpose(s) that may impact on their learning needs. Moreover, we realized that these activities might provide practical implications to a problem still unsolved in agile methods, which is cross-team knowledge sharing [17], which is an ongoing study of the authors.

From an early application of the new forms of interaction, we understood their importance in supporting the learning in the XP Laboratory course, but it was not sufficient. This first employment led us to conclude that we

[24] Image on the left was extracted from the URL: http://www.thekua.com/rant/2006/03/the-retrospective-starfish/ and presented in a tutorial at the 12th International Conference on Agile Software Development (XP 2011), entitled "Retrospectives in action" and conducted by Patrick Kua and Nick Oostvogels (http://xp2011.org/program?sid=416&o=1).

need to enhance some aspects of the new forms of interaction, and relate their positive and negative aspects to specific purposes that are presented in Table 7.

The proper selection of forms of interaction depends on project issues detected in teams, since each activity is used to achieve a certain purpose, such as (1) leveling knowledge, (2) developing insights, (3) solving problems, (4) improving or reusing solutions (course/project), and (5) raising feedback, as stated by students. Another relevant point is the time, since it is limited, we need to conduct them effectively within the time available. To accomplish this, we need to provide their explanation, voting and guidance (of what to prepare) previously at the course wiki. Also, we realized the need to create a knowledge map[25] [27] to assist the dynamics. Table 7 presents the dynamics adaptations and purposes.

Table 7: Adaptations in forms of interaction gathered from the pilot case study

Substantive Issue	Purpose	Adaptation
Mini-lectures	(1) and (2).	- Provide lectures on mandatory topics: release/iteration planning, user stories, project estimates and velocity, tests, tracking, refactoring and continuous integration. - After, provide intercalation of dynamics and mini-lectures with anticipated free topics for voting according to detected project issues.
Coding Dojo	(1), (2), (3), and (4).	- Propose feasible programming challenge.
Lightning talks	(1), (2), and (4).	- For gathering feedback on the short presentations, use scales and not the symbols of (+) and (-). - Clarify what to evaluate in short presentations and present early at the course wiki.
Brainwritting	(2), (3), and (4).	- Establish types of project issues to write in the cards, according to the class experience level. Very complex issues may not gather good contributions. - Write the cards (issues and contexts) previously with the team, for instance at team retrospective. - Coaches should also spread up to other circles.
Whole-class retrospective	(3), (4), and (5).	- Increase periodicity (every month).
Starfish diagram	(2) and (5).	- Attach it in the laboratories every month two weeks before the whole-class retrospective.
Workshops	(1), (2), (3), and (4).	- Hands-on or ludic approaches for learning XP.
Birds of a feather session	(2), (3), and (4).	- Stimulate discussion of shared interests through knowledge matrix.
Members rotation	(1), (2), (3), and (4).	- Rotate team members, when proper.
Walk the talk	(1), (2), and (4).	- Encourage an in action presentation of groups' project context, issues and challenges [27].
Role of meta-tracker	(3), (4), and (5).	- Instructor and meta-coaches should provide more support on enhancing project tracking.

5 Final Considerations

This exploratory study seeks to advance our knowledge on how to continuously improve agile methods courses of similar structure and content. It was accomplished in two stages to understand what to improve and how to apply the improvement actions in the next editions of the course through students' feedback. We diagnosed improvement actions and then, we refined our research design by applying initiatives to promote interaction between developers/students in a pilot case study.

By the qualitative data analysis of evidences from questionnaires, an agile retrospective report and a pilot case study, our results inform that improvement actions like apply diverse forms of interaction with all according to specific purposes, more detailed exposition of the subject, easy the preparation of the software development environment, and continuous students evaluation during the course can greatly facilitate learning in action. We also realized that they need to be carefully adapted and tailored during the course to reach students' learning needs.

We also included in the protocol objective periodic evaluation criteria, since the students' evaluation should not only occur by informal feedback. It should occur at the end of every month considering individual and team performances through team members' self-evaluation (grade). Also, it should consider coaches' evaluation of each

[25] Collective information on the students expertise stored in a wiki to facilitate identification of knowledge owners.

team member regarding participation and presence. And then, through faculty evaluation considering customer satisfaction grade (the customer provides a grade about each delivery and the customer approval meetings), team engagement (XP practices, teamwork, cohesiveness and self-organization), team collaborative workspace (legibility, project tracking, user stories and creativity), and software development project results during the course. In addition, we included in the protocol course and projects wiki pages, and a repository of lessons learnt from both editions.

This endeavor is part of a wider research project on the XP Laboratory course and it informs the next phases by providing important issues to consider in further studies. In the 2012 edition of the course, we intend to employ the whole process of GT through an iterative process of joint data collection and analysis within a case study to develop a substantive theory on the theme. As a next step, we will update our case study protocol with the substantive and methodological issues to engage in a definitive study to develop theories of greater scope, involving continuous improvement in agile teams/organizations and knowledge sharing among teams.

Acknowledgements

This work was supported by FAPESP project no. 2009/16354-0 and CNPQ project no.476661/2010-2. We are grateful to all students of XP Laboratory (course editions 2010 and 2011) for providing valuable information on the case in study. We would like to thank Giovanna Avalone for proofreading the paper.

References

[1] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland, and D. Thomas. *Manifesto for agile software development*. 2001. Available at: http://agilemanifesto.org/.

[2] T. Dingsoyr, T. Dybå, and N. B. Moe, *Agile Software Development - Current Research and Future Directions*. Springer, 1st edition. 2010.

[3] C. Bunse, R. L. Feldmann, and J. Dörr, "Agile Methods in Software Engineering Education". *Lecture Notes in Computer Science*, vol. 3092, pp. 284-293, 2004.

[4] H. Corbucci, A. Goldman, E. Katayama, F. Kon, C. O. Melo, and V. Santos, "Genesis and Evolution of the Agile Movement in Brazil - A Perspective from the Academia and the Industry". Accepted for publication in*: SBES is 25, Proc. of 25th Brazilian Symposium on Software Engineering*, São Paulo, Brazil, September 2011.

[5] A. Goldman, F. Kon, P. J. S. Silva, and J. Yoder, "Being extreme in the classroom: Experiences teaching XP". *Journal of the Brazilian Computer Society*, vol. 10, no. 2, pp. 5-21, November 2004.

[6] G. Melnik and F. Maurer, "Introducing Agile Methods in Learning Environments: Lessons Learned". *Lecture Notes in Computer Science*, vol. 2753, pp. 172-184, 2003.

[7] C. H. Becker, *Using eXtreme Programming in a Student Environment: A Case Study*. Master thesis, Department of Computer Science, Karlstad University, Sweden, 2010.

[8] M. Wainer, "Adaptations for Teaching Software Development with Extreme Programming: An Experience Report". *Lecture Notes in Computer Science*, vol. 2753, pp. 199-207, 2003.

[9] M. M. Müller, J. Link, R. Sand, and G. Malpohl, "Extreme Programming in Curriculum: Experiences from Academia and Industry". *Lecture Notes in Computer Science*, vol. 3092, pp. 294-302, 2004.

[10] O. Hazzan and Y. Dubinsky, "Teaching a software development methodology: the case of extreme programming". *In the Proceedings of 16th Software Engineering Education and Training (CSEE&T)*. ISSN: 1093-0175, pp. 176-184, March 2003.

[11] K. Beck, *Extreme Programming Explained: Embrace Change*. Addison-Wesley, 1st ed., 2000.

[12] K. Beck and C. Andres, *Extreme Programming Explained: Embrace Change*. Addison-Wesley, 2nd ed., 2004.

[13] K. M. Eisenhardt, "Building Theories from Case Study Research". Academy of Management Review, vol. 14, no. 4, pp. 532-550, 1989.

[14] R. Oliveira and A. Goldman, "How to build an informative workspace? An experience using data collection and feedback". Accepted for publication in the *Proceedings of the Workshop on New and Emerging Results*, Agile 2011. Salt Lake City, USA, August 2011.

[15] L. Williams, "Agile Software Development Methodologies and Practices". *Advances in Computers*, vol. 80, pp. 1-44, 2010.

[16] E. Derby and D. Larsen, *Agile Retrospectives - Making Good Teams Great*. Pragmatic Bookshelf, 2006.

[17] V. Santos and A. Goldman, "An Approach on Applying Organizational Learning in Agile Software Organizations". *In: Agile Processes in SE and XP. Lecture Notes in Business Information Processing*, vol. 77, no. 4, pp. 324-325, 2011.

[18] M. Bravo and A. Goldman, "Reinforcing the Learning of Agile Practices Using Coding Dojos". *Lecture Notes in Business Information Processing*, vol. 48, no. 4, pp. 379-380, 2010.

[19] A. Freire, F. Kon, and A. Goldman, "The 'Bootstrap' and 'Split Personality' AntiPractices - eXPeriences Teaching eXtreme Programming". *In the Proc. of the 1st Workshop on Rapid Application Development in the Brazilian Symposium on Software Quality*, pp. 73-80. June 2007.

[20] D. Sato, D. Bassi, M. Bravo, A. Goldman, and F. Kon, "Experiences tracking agile projects: an empirical study". *Journal of the Brazilian Computer Society*, vol. 12, no. 3, pp. 45-64, 2006.

[21] J. Corbin and A. C. Strauss, *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory*. Sage Publications, 3rd edition, 2007.

[22] B. G. Glaser and A. L. Strauss, *The Discovery of Grounded Theory: Strategies for Qualitative Research*. Aldine Publishing Company, Chicago, IL, USA, 1967.

[23] C. Urquhart, H. Lehmann, and M. D. Myers, "Putting the theory back into grounded theory: guidelines for grounded theory studies in information systems". *Information Systems Journal*, vol. 20, no. 25, pp. 357–381, 2010.

[24] D. T. Sato, H. Corbucci, and M. V. Bravo, "Coding dojo: An environment for learning and sharing agile practices". *In the Proceedings of the Agile Conference*, pp. 459–464, 2008.

[25] P. Sfetsos, I. Stamelos, L. Angelis, and I. Deligiannis, "An experimental investigation of personality types impact on pair effectiveness in pair programming". *Emp. Softw. Eng.*, vol. 14, no. 2, pp. 187–226, 2009.

[26] X. Meng, Y. Wang, L. Shi, and F. Wang, "A process pattern language for agile methods". *In the Proc. of the 14th Asia-Pacific Software Engineering Conference (APSEC '07)*, pp. 374–381, Washington, USA, 2007.

[27] M. Dierkes, A. B. Antal, J. Child, and I. Nonaka, *Handbook of Organizational Learning and Knowledge*. Oxford Press, 2001.

[28] R. K. Yin, *Case Study Research: Design and Methods Applied Social Research Methods*. Sage Publications, Inc., 4th edition, 2008.

[29] D. A. Schön, *The Reflective Practitioner*. BasicBooks, 1983.

[30] C. Argyris and D. A. Schön, *Organizational Learning II*. Addison-Wesley, 1996.

[31] T. H. Davenport and L. Prusak, *Working Knowledge: How Organizations Manage What They Know*. Harvard Business School Press, Cambridge, MA, 1998.

[32] I. Nonaka and H. Takeuchi, *The Knowledge-Creating Company: How Japanese Companies Create the Dynamics of Innovation*. Oxford University Press, Oxford, UK, 1995.

[33] J.M. Verner, J. Sampson, V. Tosic, N.A. Abu Bakar, and B.A. Kitchenham, "Guidelines for Industrially-Based Multiple Case Studies in Software Engineering". *In the Proceedings of the International Conference on Research Challenges in Information Science (RCIS 2009)*, pp. 313-324, April 2009.

[34] K. Beck, *Test-Driven Development by Example*. Addison Wesley. 2003.