Evaluating Collaborative Practices in Acquiring Programming Skills: Findings of a Controlled Experiment

Bernardo Estácio[*], Roberto Oliveira[†], Sabrina Marczak[*], Marcos Kalinowski[¥], Alessandro Garcia[†], Rafael Prikladnicki[*], Carlos Lucena[†]

[*]PUCRS, Porto Alegre, Brazil
{bernardo.estacio, sabrina.marczak, rafaelp}@pucrs.br
[†]PUC-Rio, Rio de Janeiro, Brazil
{rfelicio, afgarcia, lucena}@inf.puc-rio.br
[¥]Universidade Federal Fluminense, Rio de Janeiro, Brazil
kalinowski@ic.uff.br

Abstract **— [Context] Collaborative programming is achieved when two or more programmers develop software together. Pair Programming and Coding Dojo Randori are two increasingly adopted practices for collaborative programming. While the former encourages the collaboration in pairs, the latter promotes collaboration in groups. However, there is no broad understanding about the impact of these practices on the acquisition of programming skills. [Goal] In this study, we empirically compare the influence of both collaborative practices on two essential aspects of skill acquisition: motivation and learning. [Method] We conducted a controlled experiment with novice programmers applying solo programming and both collaborative practices to three different programming exercises using a crossed design. [Results] Our results showed that, while both practices outperformed solo programming, they also presented complementary benefits on acquiring programming skills. For instance, the programmers inserted less code anomalies in Coding Dojo Randori sessions than in Pair Programming sessions. On the other hand, the motivation was often considered to be stronger in the latter than in the former. [Conclusions] Our results suggest that the use of collaborative practices is particularly promising for acquiring programming skills, when programmers have little or no practical experience with software development.**

Keywords — Collaborative Programming; Pair Programming; Coding Dojo Randori; Programming Skills; Controlled Experiment.

I. INTRODUCTION

Collaborative programming consists of at least two programmers working jointly in the same algorithm or code [12]. Collaboration plays an important role in the context of acquiring programming skills, through the learning of new concepts and providing motivation for the programmers. In this context, students not only need to know how to develop software that can be easily understood by others, but they also need to learn how to develop software with others, knowing what it means working on a team [23]. In addition, in professional software development, the developer in often part of a team. Software is the result of the efforts of many individuals who contribute with different skills. Therefore, there is a need to investigate and to understand the impact of different types of collaborative practices in skill acquisition.

Collaborative programming practices mainly differ in the way the number of participants is allocated to the software development activity. While some of them encourage the collaboration in pairs [21], others promote collaboration in teams [10]. Pair Programming (PP) is a typical representative of the former, while Coding Dojo is an emerging technique for team-based collaboration. PP promotes collaboration between two programmers. Pair Programming is known as one of the main practices of Extreme Programming (XP), a widely adopted agile method [15]. Over the years, PP has been established an important role in computer higher education [3,6 21].

More recently, Coding Dojo has emerged as a team-based programming practice, providing an environment that supports social interaction and programming training [10]. There are several variants of Coding Dojo [10] and one of them is called Coding Dojo Randori (CDR). In this variant, a group of programmers gather to train software development activities. Proponents of CDR speculate that team-based collaboration would naturally promote better acquisition of programming skills than other collaborative practices [13].

However, there is no comparison about the influence of pair-based and team-based collaboration practices on the process of acquiring programming skills. Existing studies [10, 14] do not provide an analysis of key aspects on skill acquisition, i.e. motivation and learning. Most studies have either assessed each collaborative practice in isolation or focused on a particular aspect of skill acquisition. For example, PP has been studied from certain points of view, such as effectiveness in software development [27] and learning support [21]. However, such previous studies do not contrast PP with other collaborative practices, such as CDR. In addition, the study of Salleh et al. [21] considers a narrow overview of learning in existing PP empirical studies and does not take into account motivational aspects. Even worse, empirical studies have not fully assessed the impact of CDR on acquisition of programming skills, i.e in relation to learning and motivation. Some studies [14, 24] only report positive results on the learning of agile practices, such as Test-Driven Development (TDD). Even though these studies promoted a good starting point on the state-of-the-art, they fail in: (i) performing a comprehensive comparison of different collaborative practices, and (ii) analyzing aspects of each practice in relation to acquiring programming skills.

The study reported in this paper differs from the aforementioned studies by aiming at investigating the impact of different collaborative practices on the acquisition of programming skills. We selected Pair Programming and Coding Dojo Randori as representatives of pair-based and team-based practices, respectively. We have also compared these practices with solo programming in our quantitative analysis in order to better distinguish their impact on learning. Our study was conducted as a controlled experiment with novice programmers, i.e. undergraduate computer science students with limited experience in industry. The experiment adds to the previously conducted studies by offering the following main contributions:

- Providing an empirical comparative analysis based on the evaluation of both, Pair Programming and Coding Dojo Randori, regarding the acquisition of programming skills in terms of learning and motivation;
- Providing an analysis about two different levels of collaboration, pairs and groups, with respect to acquiring programming skills by novice programmers.

Both practices presented positive results in acquiring programming skills and outperformed solo programming. However, different outcomes were identified in some perspectives. Our qualitative analysis has indicated that CDR exerted a good side effect on Algorithm Design. The same qualitative analysis also revealed that the use of PP had a good effect on learning Java Syntax and Oriented-Object Programming. However, motivation was often an issue on the use of CDR. In particular, it was often not trivial to get a consensus amongst all the participants taking part of a CDR team. In relation to the motivation, Pair Programming showed more acceptance by the subjects in comparison to CDR. On the other hand, programmers inserted less code anomalies in CDR sessions than in pair.

The remainder of this paper is organized as follows: Section 2 describes the background for this research, presenting related work to ours. Section 3 describes the research methods used, presenting the settings of the experiment. Section 4 presents the results of the study while Section 5 concerns to its threats to validity. Section 6 discusses our findings. Section 7 concludes the paper with final considerations and an outline of our intended future work.

II. Background

A. Pair Programming

Pair Programming consists in two programmers working collaboratively on the same development-related activity, such as: designing an algorithm, structuring the code, or analyzing and testing a system [15,16]. In a PP session, one of the developers acts as driver and develops the code, controlling the keyboard and mouse. The other developer acts as the navigator (or observer), being responsible for reviewing the code, preventing and identifying logical and syntactical errors in the code. During a session, the pairs can switch the roles in specific time boxes [6]. PP is often related with agile practices and it has gained popularity as a primary practice of XP [15].

Several previous empirical studies aimed at exploring the effects of PP in the academia context by investigating specific aspects under programming skills. In relation to students' performance, two studies showed that PP helped programmers to better conduct their activities [6, 20]. Ramli and Fauzi [20] reported that students had a better performance in the activities in pairs than individually. McDowell et al. [6] showed two different performance metrics: code quality and the ability of students to apply the concepts taught during the course. At the end, code quality was assessed in a very subjective manner, and only the final grade obtained by the students was actually used as a parameter to evaluate the practice [6]. However, all these studies have focused on evaluating only PP as part of a computer science higher education course.

Nagappan and colleagues [19] and Carver et al. [5] showed that PP is a favorable practice for learning. Both studies reported that PP helps to reduce students' evasion in introductory programming courses. Chigona and Pollock [30] and McDowell and colleagues [6] conducted survey-based studies and showed that students were more motivated and had more satisfaction when using PP. Although these studies evaluate learning aspects, they do not focus in evaluating this variable specifically in the context of programming skills.

In the academia context, the aforementioned studies are limited to comparing pair programming with solo programming. Therefore, our assessment covers a gap of the previous studies by presenting an evaluation between two types of collaborative practices on programming skill acquisition. We also tried to gather additional evidence on how the students actually learned to program properly. To this end, we explored the impact of the collaborative practices on code anomalies [17]. Code anomalies were used to measure code quality of students' programs as they serve to indicate when the subjects have or not assimilated certain programming concepts.

B. Coding Dojo Randori

Coding Dojo consists in a collaborative practice in which a group of participants gathers together to learn and practice software development activities. Therefore, it serves the purpose of helping individuals on acquiring skills required to work on the field [10]. In the literature, Coding Dojo is often related to the learning of agile practices, such as TDD and refactoring [10].

There are several types of Coding Dojo, and one of the variants is called Coding Dojo Randori [14]. In a Randori session, a group of participants work together with the following dynamics: (i) one participant acts as the driver, (ii) another one as the navigator, and (iii) the remaining participants act as the audience that stays most of the time in silence during the pairing, paying attention to the pairs. However, the entire audience is able to participate in a coordinated way when certain events occur; for instance, if the unit tests are passed [10]. The roles of individuals are changed in rounds. Sato et al. [10] recommends that each round should last from 5 to 7 minutes. At the end of a round, the driver moves to the audience, the navigator turns into the driver and someone of the audience starts to act as a navigator [10]. Every participant acts at least once as the driver and once as the navigator.

There are only few studies empirically assessing CDR. Sato et al. [10] reported that Coding Dojo presents good results regarding the learning of TDD and Ruby programming. This study was a first attempt to evaluate CDR, but the authors did not contrast it with other collaborative practices. More recently, Da Luz et al. [24] reported a Randori experience that aimed at investigating the learning of TDD through Coding Dojo. Their results showed that the session helped learning TDD and that Pair Programming in the Randori format supported in leveling the understanding of the group. However, they focused in assessing CDR as a practice for teaching TDD.

Heinonen et al. [14] conducted Coding Dojo sessions when teaching agile methodologies as part of an undergraduate software engineering course. The survey filled by the participants presented good results in the learning of TDD. In addition, most of the students reported perceiving the sessions as a relaxing and non-competitive environment. In this study, Heinonen and colleagues [14] did not explore aspects of motivation and specific skills that CDR could affect; they only focused on the analysis of the practice in terms of teaching TDD.

Rooksby et al. [13] reported a lack of studies evaluating the effectiveness of CDR, specifically in learning and training aspects. The authors aimed to analyze the theory behind the learning process in CDR. They analyzed two CDR sessions under the perspective of reflective practice — a learning theory that states that the subjects learn by their actions during the practice [9]. They conclude that this theory offers a good way to understand the cooperative learning in CDR.

In our previous efforts, we evaluate CDR and PP practices when teaching mockup development [1]. In this study, PP presented positive results in learning, motivation and user experience between the students. CDR, on the other hand, showed good results in learning. However, students reported challenges in user experience. To the best of our knowledge, this study was the first attempt to compare two collaborative practices in Computer Science higher education. As an initial study, we did not evaluate other aspects concerning the acquisition of programming skills, specifically code programming tasks. The study reported in this paper addresses the gap of evaluating different collaborative practices in terms of acquiring programming skills.

III. Method

A. Goal

The main goal of this study is to investigate the use of Collaborative Practices — Pair Programming and Coding Dojo, specifically the Randori version — in acquiring programming skills. The detailed goal, following the GQM template [28], is described as presented in Table 1. Based on our goal definition, we derived two specific research questions: one concerning learning (RQ1) and one concerning motivation (RQ2). The questions are presented and discussed below.

TABLE I. Detailed GQM template of the experiment

To Analyze	Coding Dojo and Pair Programming
For the purpose of	Evaluation
With respect to	Acquiring Programming Skills (Learning and Motivation)
From the point of view of	Researchers
In the context of	Novice programmers

RQ1. *How do collaborative practices affect the learning of programming skills by novice programmers?*

We divided RQ1 in two sub questions, the first concerning qualitative data on the perception of the novice programmers and the second concerning quantitative data gathered from the produced source code.

RQ1.1. *How do collaborative practices affect the perception of novice programmers on their learning of programming skills?*

To address this research question, we analyzed qualitative data collected through an experiment follow-up questionnaire. The questionnaire was built based on the constructs proposed by Wangenheim et al. [7]. The use of these constructs enables us to evaluate the effect on learning and on motivation.

RQ1.2. *How do collaborative practices affect the learning of programming skills by novice programmers in practice?*

For this research question we adopted the number of code anomalies inserted in practice during programming exercises to analyze the learning of specific skills from a quantitative perspective. The presence of an anomaly would indicate the subjects did not assimilate a certain programming skill. We included Solo Programming as a control group to support the evaluation of the results. The following *null* and alternative hypotheses were formulated for this research question.

H_{00}. Collaborative practices do not affect the learning of programming skills by novice programmers (inserted code anomalies), when compared to solo programming.

$H_{A1.1.}$ PP improves the learning of programming skills by novice programmers (reduces the number of inserted code anomalies), when compared to solo programming.

$H_{A1.2}$ Coding Dojo Randori improves the learning of programming skills by novice programmers (reduces the number of inserted code anomalies), when compared to solo programming.

$H_{A1.3}$ Coding Dojo Randori improves the learning of programming skills by novice programmers (reduces the number of inserted code anomalies), when compared to pair programming.

RQ2. *How collaborative practices affect motivation of the novice programmers?*

As done for RQ1.1, to address this research question, we analyzed qualitative data collected through the experiment follow-up questionnaire.

B. Subjects

The study was conducted in an Agile Software Development course, called Software Kaizen [2]. This course provides undergraduate Computer Science students an immersion of four months in real projects with industry. Fourteen intern students participated in the experiment. In order to participate in the study, all the students signed a consent form. They also filled out a characterization form with objective questions to inform us about their expertise in the topics related to the study: (a) their experience in programming; (b) their expertise in Java; (c) their expertise with PP; and (d) their expertise with Coding Dojo.

We collected the data characterization form from each student and ranked it into: none (N), low (L), medium (M), and high (H) experience for each expertise topic. For instance, regarding programming and Java expertise, the subject was characterized as having: (a) No experience, if she never had contact with the Java language; (b) Low experience, if she had contact with programming only in the classes or reading a support material; (c) Medium experience if she had contact with programming in an academic project; or (d) High experience if she had experience in the industry. Similarly, the expertise for PP and Coding Dojo was assigned according to the number of sessions in which the subject had worked in such activities: (a) No experience; (b) Low experience: 1 session; (c) Medium experience: more than 1 to less than 4 sessions; and (d) High experience: more than 4 sessions.

After characterizing the participants' experience, to mitigate threats to validity concerning the distribution of subjects between the groups we applied the principles of balancing, blocking and random assignment [8]. For balancing we tried to create equally-sized groups. However, we had to create one of the groups with 6 subjects because we aimed to evaluate the interaction of programmers in a large group within a Coding Dojo session. Concerning blocking, we mean that we avoided that one team had more experienced subjects than the other in order to avoid biased results of a team performing better in the assigned tasks. Finally, subjects of equal experience were randomly assigned to the groups. Table 2 shows each of the defined groups and the expertise of each of its members, we highlighted with grey-tone the most experienced and left without color the less experienced in each group.

TABLE II. EXPERTISE PER PARTICIPANT IN EACH GROUP

Group	Id	Programming	Java	Pair Programming	Coding Dojo
1	Participant 1	High	Low	Medium	Medium
	Participant 2	Medium	Medium	Medium	Low
	Participant 3	Low	Low	Low	Low
	Participant 4	Low	Low	Low	Low
	Participant 5	Medium	Low	Low	Low
	Participant 6	Low	Low	Medium	Medium
2	Participant 7	Medium	Medium	Low	Low
	Participant 8	Low	Low	Medium	Medium
	Participant 9	Medium	Medium	Low	Low
	Participant 10	High	Medium	Medium	Low
3	Participant 11	High	Medium	Medium	Low
	Participant 12	Low	Low	Low	Low
	Participant13	Medium	Low	Low	Low
	Participant 14	High	Medium	Medium	Medium

C. Experiment Design

We planned our experiment design to include one factor with three treatments (Solo Programming, Pair Programming and Coding Dojo) and three different tasks (different programming exercises, objects of the study). We adopted a crossed design in order to enable all the treatments to be applied to all the tasks. It also helped to mitigate threats to validity of our experiment concerning: (i) differences among experimental tasks (the influence of the provided exercises in the results), and (ii) the fact that one task could favor a specific treatment. This design also helps to isolate the learning effect, given that each group will apply the practices in a different sequence. It is noteworthy that the principles applied to distribute the subjects between the groups still enable comparing the results for each individual exercise. The crossed design is shown in Figure 1.

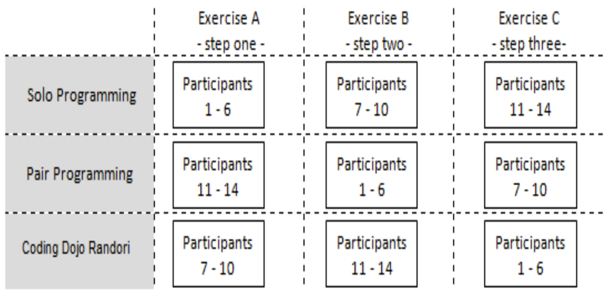

Fig. 1. Configuration of the crossed design in the experiment.

D. Procedures and Materials

Concerning the procedures and materials [22], in the first step of the experiment (cf. Figure 1) exercise "A" was applied to participants 1 to 6 with the solo programming treatment, to participants 11 to 14 with the PP treatment, and to participants 7 to 10 with the Coding Dojo treatment. Thereafter, in the second step, exercise "B" was applied to participants 7 to 10 with the solo programming treatment, to participants from 1 to 6 with the PP treatment, and to participants 11 to 14 with the Coding Dojo treatment. Finally, exercise "C" was applied to participants 11 to 14 with the solo programming treatment, to participants 7 to 10 with the PP treatment, and to participants 1 to 6 with the Coding Dojo treatment. The formation of the pairs in PP sessions within each group was similar to how it was performed by Braught et al. [11], pairing experienced subjects with one less experienced subjects. In the CDR session, the sequence of the pairs was determined by the convenience of the participants.

Each programming exercise was complex [22]. For example, in the exercise "A", the participants had to develop an application to simulate Animal Guessing game. The main objective of this application was to implement the interaction between an interviewer and a respondent. In the exercise "B", the participants had to develop some features of an Automated Teller Machine (ATM). Finally, in the exercise "C", they developed an application to control a bookstore inventory. Subjects should have to implement at least ten classes as a constraint. In order to realize the exercise tasks, the developers should explore programming logic and key mechanisms of OO programming, such as inheritance and polymorphism. The study was conducted online (simultaneously and observed by researchers) and lasted approximately two hours for each group. They worked on their tasks simultaneously in different rooms. At the end of the study, each participant answered a follow-up questionnaire and sent the implemented code to the researchers. In order to identify the quality of the software developed by the students, we evaluate and analyze two complementary strategies for the identification of code anomalies [17]: (i) manual inspection, and (ii) inspection with the aid of a tool for semi-automatic identification of anomalies [18]. The combination of these two strategies generated a list of existing anomalies. We took into consideration the anomalies documented in the Fowler's catalogue [17]. Each of these anomalies is an indicator that certain programming skills are not assimilated. For instance, the presence of a God Class or a Feature Envy [17] reveals that the subjects did not master the proper use of one or more object-oriented programming concepts, such as classes and encapsulation. The manual inspection was conducted by two co-authors of this paper. This inspection also enabled us to identify the code anomalies that were not properly detected by the employed tool [18]. The final list of code anomalies was checked by a second co-author of this paper, and the results were discussed with the other paper co-authors in order to address discrepancies.

The follow-up questionnaire aimed at capturing information concerning learning (RQ1.1) and motivation (RQ2). The questionnaire was adapted from Wangenheim et al. [7]. The dimensions and respective arguments that we have adapted were listed in Table 3. The study was executed in a different context but has a specific framework to assess both learning and motivation. The follow-up questionnaire consists of: (i) 5 fixed questions concerning motivation and learning, and (ii) 5 dimensions on a Likert-scale for each of these questions, with response alternatives ranging from strongly disagree (−2) to strongly agree (2) (being −2 strong disagree, −1 disagree, 0 neutral, 1 agree, 2 strong agree).Regarding learning, we also added questions to elicit the perceived knowledge level before and after the practice in respect to the concepts taught and in accordance to Bloom's taxonomy [4]. In order to answer this questionnaire, the students should provide grades ranging from 1 to 5 to each question based on three categories of programming skills: Java Syntax, Oriented-Object Programming and Algorithm Design. These grades were provided before and after the programming exercises. We also customized the follow-up questionnaire, adding two open questions so that students could explain perceived benefits and disadvantages about PP and Coding Dojo.

TABLE III. QUESTIONS LIKERT-SCALE ADAPTED FROM WANGENHEIM ET AL. [7]

Motivation		
Attention	There was something interesting in the practice that got my attention.	Adapted
Relevance Confidence	The practice dynamics suits well my way of learning.	Adapted
	As I worked on the practice, I felt confident that I was learning.	Adapted
Learning		
Long-term learning	The experience with the practice will improve my performance in future projects.	Adapted

IV. RESULTS

We analyzed the quantitative data (Section 3A) in order to test the hypotheses described in RQ1.2. This sub question in concerned with learning in terms of the quality of the developed source code. The quality was measured in terms of the number of anomalies inserted in the code during the exercises (Section 3D). On the other hand, the qualitative data (Section 4B) was collected and analyzed to answer both RQ1.1 and RQ2. The qualitative data was gathered from the follow-up questionnaires (Section 3D).

A. *Quantitative Analysis*

We analyzed the answers regarding the types of code anomalies reported by Fowler [17]. We also applied statistical analyses on the quantitative data on the number of inserted code anomalies obtained from the developed code. Such statistical analyses were carried out with support of the R tool [25]. This tool supports applying the statistical test considered in this study: Wilcoxon-Mann-Whitney test [26]. This test is a nonparametric test of the null hypothesis that two samples come from the same population against an alternative hypothesis. We applied it to the values associated with the produced anomaly instances. We chose this nonparametric test for not requiring a normalized and homoscedastic distribution of the data and for accepting different sample sizes for the analysis.

Figure 2 presents the average number of code anomalies inserted by each practice in each of the three exercises. Therefore, the columns highlighted in black represent the results of solo programming, in light gray represent the results of pair programming in dark gray represent the results of coding dojo.

Comparing solo programming with PP, we verified that in the exercise A, solo programmers inserted an average number of 4.83 anomalies while pairs inserted an average number of 2 anomalies. So, solo programmers inserted 58.6% more anomalies than pairs. In this exercise, when applying the Wilcoxon-Mann-Whitney test the alternative hypothesis HA1.1 is supported with confidence level of 94.3% (p-value 0.057). In exercise B, solo programmers inserted 87.2% more anomalies, also supporting the alternative hypothesis, this time with a confidence of 85.8% (p-value 0.142). Finally, in the exercise C, solo programmers inserted 47.5% more anomalies than pairs. For this exercise, the alternative hypothesis was supported, with a confidence level of 95.8% (p-value 0.042).

Although there is some preliminary indication supporting HA1.1, replications should be performed in the future with more subjects in order to enable achieving higher confidence levels. A potential explanation for this preliminary indication is that coding by pairs might allow programmers to discuss and share ideas, thus complementing the knowledge of each other and consequently coding software with less structural problems.

Comparing solo programming with CDR, we verified that in the exercise A, solo programmers inserted an average number of 4.83 anomalies while in Coding Dojo Randori session they inserted an average number of 4.75 anomalies. Therefore, the solo programmers inserted 1.6% more anomalies than groups in the CDR session. This result does not support the alternative hypothesis because p-value is 0.490. In the exercise B, this percentage was even higher, i.e., solo programmers inserted 92.24% more anomalies. Supporting alternative hypothesis with confidence level 98,6% (p-value 0.014). Finally, in the exercise C, solo programmers inserted 80.92% more anomalies than groups in the CDR session. Supporting alternative hypothesis with confidence level 99,6% (p-value 0.004).

A possible reason for this superior result of Coding Dojo is this practice's dynamics, which encourages programmers to participate collaboratively.

With the comparison between PP and CDR, it was possible to observe in the exercise A, that pairs inserted an average number of 2 anomalies while CDR teams inserted an average number of 4,75 anomalies. Therefore, the programmers in pairs inserted 57.9% less anomalies than those working in CDR session. For this exercise, the alternative hypothesis was not confirmed as the p-value is 1.0.

However, in exercises B and C, pairs inserted more anomalies. In the case of exercise B, they inserted 62.69% more anomalies. Supporting alternative hypothesis with confidence level 73,4% (p-value 0.266). Finally, in exercise C, they inserted 63.63% more than their counterparts in the CDR session. Also Supporting alternative hypothesis with confidence level 99,6% (p-value 0.004). A possible reason for this superior result of the CDR is the collaboration by combining knowledge, experience, competence and efforts of the group to achieve their goals.

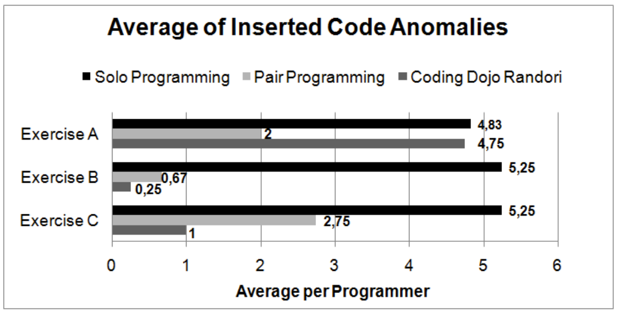

Fig. 2. Comparing the average of inserted code anomalies

B. *Qualitative analysis*

We have gathered qualitative data in the Likert-scale questions from the follow-up questionnaire.

1) *Learning*

In relation to RQ1, the majority of the participants also expressed that they believe that both practices contributed positively to their long-term learning. Figure 2 indicates this outcome for both Pair Programming (in the left side) and Coding Dojo (right side). Additionally, the students indicated that these practices could be useful in a real working environment. In addition, they reported that CDR provided initial knowledge on how they can make decisions in a project, trough the discussion of specific points in the source code.

In relation to learning based on Bloom's Taxonomy [4], we use a scale between 1-5 to measure skills of the students in each practice. As shown in Figures 4 and 5, we perceived a significant increase of knowledge with respect to all three categories of skills: Java Syntax, Object-Oriented Programming and Algorithm Design by the code produced.

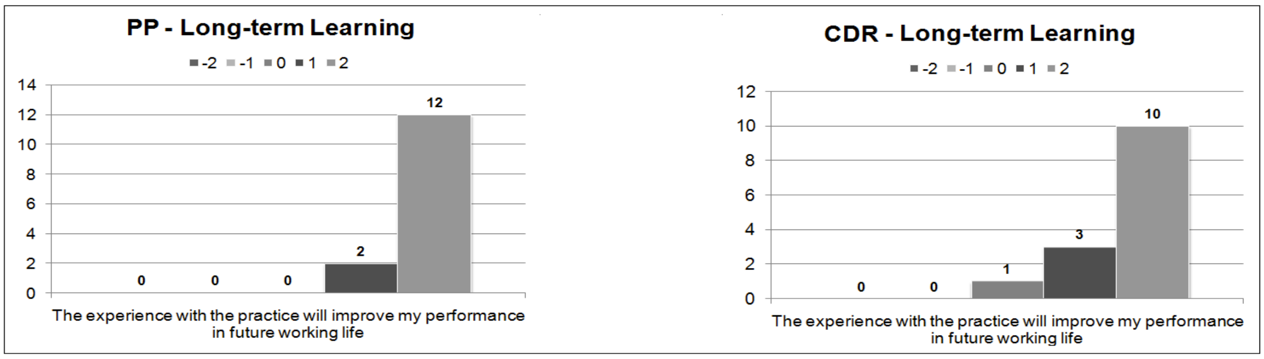

Fig. 3. Frequency diagram about Motivation in Pair Programming and Coding Dojo Randori.

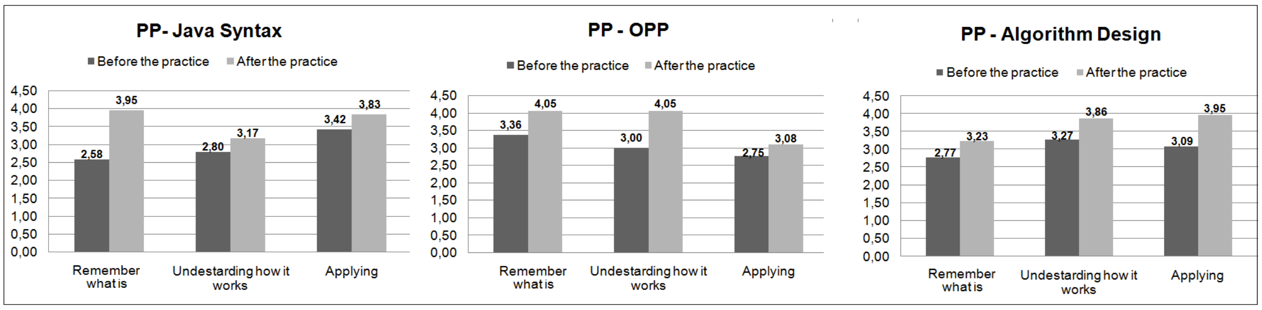

Fig. 4. Grades of Bloom Taxonomy's in Pair Programming [4].

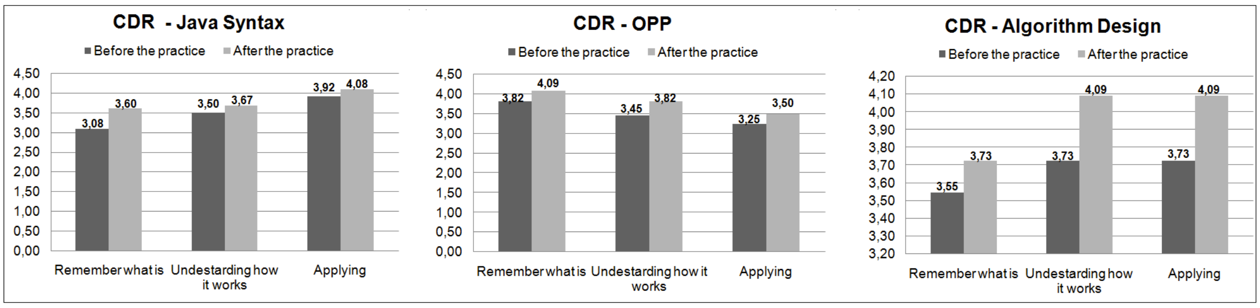

Fig. 5. Grades of Bloom Taxonomy's in Coding Dojo Randori [4].

2) Motivation

As far as the Motivation of RQ2 is concerned, students perceived a positive contribution of Pair Programming when developing the programming tasks. This information is based on the responses related to PP and CDR obtained through the Likert-Scale for each argument of the follow-up questionnaire. Figure 6 indicates the answers for both practices.

When we analyzed individually the dimensions, for instance Attention, both practices presented positive results. The main aspect that got the attention of the participants in both practices was the interaction between the subjects. Participant 3 reported in relation of PP: *"I like to discuss the solution and design of the task with my peer and this helped to get my attention through the session time"*.

Regarding the Relevance dimension, PP presented positive results with most of the students in relation to acceptance of the practice, but CDR was not widely accepted. Four participants reported that the practice did not suit with their way of learning. In PP, Participant 4 reported: "*Pair Programming is better to get a consensus than Coding Dojo; when we were in a Coding Dojo session, it was very difficult to agree with everyone else"*.

Concerning the Confidence dimension, the items related to understanding and ease of use of the practice, both PP and CDR presented positive results. PP presented the most positive results about the students' impression of confidence in learning. However, CDR also presented positive results. Participant 4 reported: "*In the Coding Dojo Randori session experienced programmers could support us during all the session, so we have more confidence in doing the right things"*.

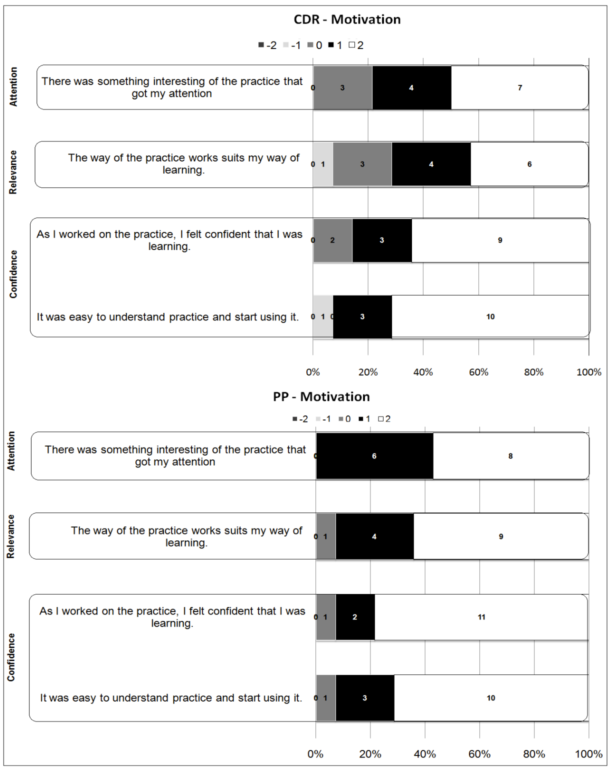

Fig. 6. Frequency diagram about Motivation in Pair Programming and Coding Dojo Randori.

V. Threats To Validity

In this section we discuss the potential threats that are relevant for our study and how they are addressed.

A. *Construct Validity*

Construct validity concerns the relationship between theory and the observation, i.e., if the treatment correctly reflects the cause construct and if the result correctly reflects the effect construct [8]. In this study we evaluated the effect on learning and motivation by following the constructs proposed by Wangenheim et al. [7]. Also, for code quality, we evaluate the effect by measuring the inserted anomalies, as defined by Fowler [17]. Thus, we believe that the treatment (the collaborative practice was the only difference in the treatments) does represent the cause and that the metrics do represent the effect. Still we are aware that there are some threats to construct validity that are difficult to control, such as subjects behaving differently when being observed and thus the potential influence of the presence of the researcher in the experiment.

B. *Internal Validity*

Internal validity concerns a causal relationship between the experiment treatment and the observed results, without the influence of potential confounding factors that are not controlled or measured [8]. In this study, we considered the following main threats to the internal validity: (a) difference among subjects related to experience and ability (e.g., programming and Java knowledge, pair programming and coding dojo expertise), and (b) differences among experimental tasks (the provided exercises could have influenced the results). To mitigate the threat of differences among subjects we applied the design principles of balancing, blocking and random assignment, as suggested by Wohlin et al. [8]. Regarding the influence of the experimental tasks (exercises), we applied the control action of applying a crossed design, in which all treatments were applied to all tasks by independent groups. Leading us to three independent trials. It is noteworthy that the principles applied to distribute the subjects between the groups still enable comparing the results for each individual task. This design also helped to isolate the learning effect, given that each group applied the practices in a different sequence.

C. *External Validity*

External validity refers to the generalization of the results to a larger population or to other environments and to populations that differ from the one studied. This study presents several threats to external validity (e.g., related to the representatives of the subjects, the representativeness of the provided tasks, and the specific experimental environment). All the students in the study came from the same course. For this reason, the data extracted from this study presents important results related to motivation, learning and code quality, but cannot be generalized at this time. Therefore, as an initial study on this subject, we do not raise any external validity claims and ask for replications (e.g., with different subjects, using different tasks, in different environments) to allow further generalizing the results.

D. *Conclusion Validity*

Finally, conclusion validity (or statistical conclusion validity) regards the relation between the treatment and the results in terms of statistical significance [8]. In this study, the major limitation is the small sample size which led us to use high alpha values and thus to a lower confidence level of the achieved results. Thus, more studies and replications are also needed to improve the conclusion validity.

VI. Discussion

Collaboration plays an important role in the context of acquiring programming skills, through the learning of new concepts and providing motivation for the programmers. In this paper, we investigate the impact of collaborative practices on the acquisition of programming skills. We selected Pair Programming and Coding Dojo Randori as representatives of pair-based and team-based practices, respectively. Overall, the results showed positive results for both practices in acquiring programming skills. However, the results were not consistently the same from all perspectives, revealing peculiarities through the use of each practice. We observed that there are many opportunities to investigate the adoption of collaborative practices in acquiring programming skills.

When we quantitatively analyzed the effect on learning, we observed both practices present better results than solo programming. This analysis enabled us to understand how the students were acquiring (or not) the programming skills by computing the anomalies inserted in the source code. Each of the observed anomalies were a direct consequence of the students not grasping a certain programming concept, construct or principle. This result led to identify that collaborative practices, independently of different types of collaboration (PP or CDR), positively impact on the acquisition of programming skills. However, the results we obtained showed a variation in the number of anomalies inserted in the three exercises by each collaborative practice. Consequently, future work should further investigate in what occasions we must apply PP or CDR in the development of a specific task.

When we analyzed the qualitative data concerning learning, we observed that both practices support the students in the three skills that we analyzed. Both practices showed good results that were perceived through the grades of [4] Taxonomy's in all skills. CDR has received higher grades than PP in the three skills, specifically in Algorithm Design, while in PP we identified a better evolution (observing the range before and after the using of the practice) in grades related with Java Syntax and Oriented-Object Programming when we compared with the evolution in CDR. In addition, we observed that in CDR there was more discussion on how the team could improve the code. These discussions sometimes did not reach consensus, but helped in the understanding the problem and designing the solution. But PP showed to be a practice more consolidated than CDR, the subjects reported that the pair-based approach affects more the working life than CDR.

In relation to the Motivation, the perception of the participants through the questionnaires' results shows that CDR presented challenges in relation to the consensus on decision making. However, some students reported that the lack of a consensus encourages to understand further the concepts needed to perform the task in question. On the other hand, some students said that the audience helped them to feel confident in learning with the programming task. This situation were by we anticipated, therefore, to minimize this impact we limit the audience in the CDR session. So, we did two sessions with four students and one with six.

Qualitative and quantitative data gave some important directions on the research of different collaborative practices. For instance, in CDR we perceived variables that could affect the practice, such as the type of task and the number of participants in audience. In addition, although PP is more consolidated in the literature [21], we have also identified some opportunities of research, such as the formation of the pairs (we have followed with a more experienced programmer with a less experienced one) and time to switch the roles (if this could impact in the activity).

VII. Conclusion and Future Work

According to Basili [29], we must experiment with practices to see how and when they really work, to understand their limits, and to understand how to improve them. In this paper, we presented a controlled experiment where we investigated the use of Pair Programming and Coding Dojo Randori practices in relation to acquiring programming skills. The context of this study was in an Agile Software Development course with novice programmers. Through a qualitative and quantitative analysis, both practices presented positive results on key aspects of acquiring programming skills, i.e. learning and motivation. However, the analysis revealed important differences in the outcomes of each aspect. As far as learning is concerned, the quantitative analysis showed that both practices outperformed solo programming. In addition, the programmers inserted less code anomalies in CDR sessions than in PP sessions.

Qualitative data about learning revealed that both practices are complementary, i.e. PP and CDR yielded better results for specific programming skills. For instance, CDR outperformed PP in terms of Algorithm Design, while Pair Programming outperformed CDR in terms of Java Syntax and OPP. As far as the motivation is concerned, the findings show that Pair Programming is a more consolidated practice in this aspect. Coding Dojo Randori presented challenges through the lack of consensus.

Future steps in this work involve the planning and execution of new empirical studies in order to evaluate CDR and PP in relation to specific variables. Examples of these variables include team size, the impact of specific programming tasks, and the formation of the pairs. We also want to evaluate the impact of CDR and PP on other software development skills, such as refactoring and environment configuration. We expect that our findings can be useful for higher education teachers and students by providing first insights for the adoption of the collaborative practices on teaching and learning programming.

Acknowledgment

The authors acknowledge financial support in this research from CNPq (processes 309000/2012-2 and 460627/2014-7), FAPERGS, and the research agreement between PUCRS and ThoughtWorks.

References

[1] B. Estácio, N. Valentim, L. Rivero, T. Conte, R Prikladnicki,"Evaluating the Use of Pair Programming and Coding Dojo in Teaching Mockups Development: An Empirical Study," Hawaii International Conference on System Sciences, HICSS, 2015, pp 5084-5083.

[2] B. Estacio, R. Prikladnicki, M. Mora, G. Notari, P. Caroli, A. Olchik, "Software Kaizen: Using Agile to Form High-Perfomance Software Development Teams," Agile Conference (AGILE), 2014 , pp.1-10, July 28 2014-Aug. 1 2014.

[3] B. Simon and B. Hanks, "First-year students' impressions of pair programming in CS1".J. Educ. Resour. Comput., vol. 7-4, pp. 1-20, ACM, 2008.

[4] B.S. Bloom, "Taxonomy of Educational Objectives: The Classification of Educational Goals,": Handbook I. Cognitive Domain, Longmans, 1956

[5] C. Carver et al., "Increased Retention of Early Computer Science and Software Engineering Students Using Pair Programming,". In: Software Engineering Education & Training, 2007, pp.115-122.

[6] C. Mcdowell, "The effects of pairprogrammingon performance in an introductory programming course". In: SIGCSE technical symposium on Computer science education, 2002, pp. 38–42.

[7] C. Wangenheim, R. Savi, R. Borgatto, "SCRUMIA—An educational game for teaching SCRUM in computing courses," Journal of Systems and Software, vol. 86 - 10, Pages 2675-268, 2013.

[8] C. Wohlin, P. Runeson, M. Host, M. Ohlsson, B. Regnell, and A. Wessl, "Experimentation in software engineering: an introduction, Kluwer Academic Publishers, 2012.

[9] D.A., Schön, "Educating the Reflective Practitioner: Toward a New Design for Teaching and Learning in the Professions," Jossey-Bass ,1990.

[10] D.T Sato, H. Corbucci and M.V Bravo, "Coding Dojo: An Environment for Learning and Sharing Agile Practices," Agile Conference, (Toronto, Canada), pp.459-464, IEEE, 2008.

[11] G. Braught, K. MacCormick and T. Wahls, "The Benefits of Pairing by Ability," ..*In Proceedings of*SIGSE, 2010, 249-253.

[12] J. Nosek, "The case for collaborative programming. Commun," ACM 41, 3 (March 1998).

[13] J. Rooksby, J. Hunt. And X. Wang,"The Theory and Practice of Randori Coding Dojos," International Agile Conference, XP, 2014, pp 251-259.

[14] K. Heinonen, K. Hirvikoski, Matti Luukkainen, and Arto Vihavainen, "Learning agile software engineering practices using coding dojo,". Proceedings of the 14th annual Conference on Information Technology Education (SIGITE '13). , (New York, USA) pp. 97-102, 2013.

[15] K. Beck and C. Andres, "Extreme Programming Explained: Embrace Chance". Boston: Addison-Wesley, 2004.

[16] L. Williams et al. "Strengthening the Case for Pair Programming". In IEEE Software, 17, 4, Jul. 2000, pp. 19-25.

[17] M. Fowler. et al, "Refactoring: Improving the Design of Existing Code"., Addison-Wesley Professional, 1999.

[18] E. Murphy-Hill. and T.Black, Seven Habits of a Highly Effective Smell Detector", In Proceedings of RSˇSE, 2008, 36-40.

[19] N. Nagappan et al., "Improving the CS1 experience with pair programming". In: SIGCSE Bull, 2003, pp. 359-362.

[20] N. Ramli and S. Fauzi, "The effects of pair programming in programming language subject". In: International Symposium on Information Technology, 2008, pp.1-4.

[21] N. Salleh, E. Mendes and J. Grundy, "Empirical studies of pair programming for CS/SE teaching in higher education: A systematic literature review," IEEE Transactions on Software Engineering, vol.37-4, pp. 509–525, 2011.

[22] Open-Archives. Available at: http://www.inf.puc-rio.br/~rfelicio/pages/experiment02.html, April 2015

[23] R. Arora and S. Goel., "Learning to Write Programs with Others: Collaborative Quadruple Programming," IEEE 25th Conference on Software Engineering Education and Training (CSEE&T), (Nanjing, China) , pp.32-41, IEEE, 2012.

[24] R.B. Da Luz, A.G Neto and R. Noronha, "Teaching TDD, the Coding Dojo Style," International Conference Advanced Learning Technologies (ICALT), (Beijing, China),pp.371-375, IEEE, 2013.

[25] R Tool. Available in: <http://www.r-project.org/>. Access in March, 2015.

[26] S. Siegel, and J. Castellan. "Nonparametric Statistics for the Behavioral Sciences", 2nd Edition. Mc-Grawl-Hill International Editions, 1988.

[27] T. Dyba, E. Arisholm, D.I.K Sjoberg, J.E Hannay and F. Shull, "Are Two Heads Better than One? On the Effectiveness of Pair Programming," *Software, IEEE* , vol.24, no.6, pp.12,15, Nov.-Dec. 2007.

[28] VR . Basili, V.R., C. Caldiera, H.D. Rombach, 'Goal Question Metric Paradigm', Encyclopedia of Software Engineering (Marciniak, J.J., editor), Volume 1, John Wiley & Sons, 1994, pp. 528-532.

[29] V. R. Basili, *"The role of experimentation in software engineering: past, current, and future"*. IEEE International Conference on Software Engineering (ICSE), pp. 442-449, 1996.

[30] W. Chigona and M. Pollock "Pair programming for information systems students new to programming: Students' experiences and teachers' challenges". In: Portland International Conference on Management of Engineering & Technology, 2008, pp.1587-1594.