Information Systems Development with Pair Programming: An Academic Quasi-Experiment

Eduardo Fernandes, Fischer Ferreira, João Antônio Netto, Eduardo Figueiredo
Software Engineering Laboratory (LabSoft), Department of Computer Science
Federal University of Minas Gerais (UFMG)
Belo Horizonte, Minas Gerais, Brazil
{eduardofernandes, fischerjf, joaoantonio, figueiredo}@dcc.ufmg.br

ABSTRACT

Pair Programming is a development technique in which two programmers collaborate to conduct the same development task. The use of this technique in information systems development may support many activities, such as code inspection and software integration. Studies have investigated the advantages and drawbacks of pair programming in both industrial and academic context. However, with respect to academic research, the majority of studies investigate this technique in European or North American educational institutions. Considering that some social and geographic factors may impact on the application and efficiency of agile methods such as pair programming, we lack an evaluation of this programming practice in the context of Brazilian students. In this paper, we discuss the findings of three one-hour quasi-experiments conducted with 55 undergraduate and graduate students to assess pair programming in the development of tasks to implement an information system. These participants are students enrolled in Information Systems and related courses of two Brazilian institutions. For the experiment, we divided each class in two groups: one group for solo programming and the other for pair programming. As a result, we observed that participants developing tasks in pairs presented lower rates of time spent and difficulty faced to complete development tasks when compared with solo programming participants. However, we did not observe a significant increase on the correctness in tasks developed by both experiment groups: paired and solo programmers. Finally, we conducted an analysis of participant feedback regarding other advantages of using pair programming in systems development.

Categories and Subject Descriptors

H.0 [**Information Systems**]: General. D.2.5 [**Software Engineering**]: Testing and Debugging – *code inspections and walk-throughs*. D.2.9 [**Software Engineering**]: Management – *programming teams*.

General Terms

Experimentation, human factors, management, verification.

Keywords

Information systems, software development, pair programming, academic context, superior education.

1. INTRODUCTION

Pair Programming (PP) is a software development technique, related to Agile Methods, in which two programmers collaborate to conduct the same development task [2][9]. One of the programmers assumes the control of computational resources and writes source code. The other programmer is responsible for assisting the partner, verifying the source code during implementation to identify flaws or mistakes, for instance [8]. This technique may also contribute for discussion and brainstorming between partners to solve development tasks [5].

PP has been successfully applied in industry [2][23] aiming to increase software quality, foster team collaboration, introduce junior developers in a development team [19], and many other contexts [10]. There are some investigation of pair programming in academic context. For instance, previous work [12][24] has assessed the impact of PP in the performance of undergraduate students. One of the main positive impacts of PP in academia is the knowledge sharing that can support undergraduate students in programming courses [14]. However, we still lack a deeper understanding regarding the benefits of PP in educational environments (in general) and in Brazilian contexts (in particular).

To address this limitation, this study presents and discusses the findings of three one-hour quasi-experiments comparing PP with solo programming in the context of two Brazilian universities. For this purpose, we collected data regarding time spent, difficulty level, correctness, and feedback with respect to five development tasks conducted by experiment participants. We aim to identify advantages and drawbacks of using PP when compared with solo programming in the development of information systems. We designed tasks to be conducted in a one-hour experiment, using Java programming language, and on a toy information system for improvement or implementation of new features.

The experiments were conducted in three university courses of two educational institutions; one of the courses has post-graduate level. For the courses, undergraduate and graduate students were randomly divided in two groups, one to use PP and the other to use solo programming during the experiment. Each participant (or pair of participants) conducted five development tasks in a laboratory, using computers with support material, such as a pre-installed IDE, source code of an information system, and its documentation.

As a result, we observed significantly lower rates of time spent and difficulty faced by participants who developed the proposed tasks in pairs, when compared with solo programmers. However, we did not observe a significant increase of correctness in the

source code developed by pair programmers in comparison with programmers from the solo group.

The remainder of this paper is organized as follows. Section 2 presents a background with relevant concepts to support the understanding of our study. Section 3 describes the experimental design, including the experiment scope and investigated hypotheses. Section 4 presents and discusses our main research results regarding the three applied quasi-experiments. Section 5 presents related work with respect to the investigation of PP in academic and industrial contexts. Section 6 discusses some threats to the validity of our study. Finally, Section 7 concludes this paper and suggests future work.

2. BACKGROUND

Agile methods are a collection of development and management techniques to support lightweight and fast software development [9]. Agile methods are composed by various techniques with different goals, such as prioritizing high software reliability [13], simple documentation for small development teams [7], and client satisfaction with respect to software product fast deliver [1]. Some of the agile principles are continuous development of software requirements, promotion of developers' motivation and cooperation, technical improvements, appropriate software design, and software quality [11].

Agile software development has been proposed in response to limitations of traditional plan-oriented software processes, such as the Waterfall model [9]. The Agile Manifesto[1] focus on four key statements: individuals instead of processes and tools, functional software over extensive documentation, collaboration between stakeholders and development team, and responding to change. This manifesto also compiles the 12 principles of agile software development that cover several software development practices. Fowler and Highsmith [11] present agile methods as a new development approach covering various programming techniques and practices. Typical examples of agile methods include Extreme Programming (XP) [3], Scrum [4], and Test-Driven Development (TDD) [13].

XP is a collection of practices to support agile software development that covers technical aspects, such as source code implementation, software design, and testing. XP prescribes short development cycles and flexible scheduling [3]. Scrum is another agile method to support development management. Scrum aims to minimize the overload of the traditional central control of development teams, in a decentralized, iterative and incremental fashion [21]. The development process is distributed to teams under inspection. Scrum is recommended to complex and large software systems, for instance [22].

TDD applies a technique in which software testing comes before source code development [6]. It fits to critical and large systems that require special attention to flaw detection, because it was design to support increase of software quality by minimized occurrences of bugs, for instance [17]. Its steps consists of (i) writing unit tests to be run before a feature implementation, (ii) developing of the aimed features, and (iii) successive testing until the feature implementation is in accordance with the unit test [13].

In this study, we are interested on evaluating a specific agile practice: Pair Programming (PP). PP is a software development practice in which two programmers conduct the same development task [2]. One of the programmers is responsible to control the computational resources and write source code. The other programmer is responsible for assisting the partner in source code verification and validation, discussion, and other contributions [8]. PP is a practice particularly recommended by XP [18]. This practice have been investigated both in industry [2][23] and academia [12][24].

[1] http://www.agilemanifesto.org/principles.html/

3. EXPERIMENTAL DESIGN

This section describes the experimental design of our study. In Section 3.1, we present the main goal and hypotheses that guided this study. Section 3.2 describes the participant set before data set reduction. Section 3.3 presents the artifacts we designed to support our experiments. Finally, Section 3.4 describes the steps of the experiment.

3.1 Goal and Hypothesis

In this study, we aim to evaluate the use of PP in academic context when compared with solo programming, with respect to the development of information systems. For this purpose, we conducted three quasi-experiments in three Information Systems-related courses in two educational institutions: Federal University of Minas Gerais (UFMG) and University of Itaúna (UIT). Two courses are undergraduate level (Software Engineering) and one is post-graduated level (Empirical Software Engineering).

Our main goal is to identify advantages and drawbacks regarding the use of PP by undergraduate and graduate students in the development of programming tasks. We are also interested in the participant feedback about the experiment.

The experiment goal, based on the Goal-Question-Metric (GQM) method [25], is: *Analyze the use of Pair Programming (PP), from the purpose of performance evaluation when compared with solo programming, with respect to time spent, difficulty level, and correctness of developed tasks, from the point of view of developers, in the context of Brazilian students*.

To support our empirical study, we conceived the following hypotheses.

H1. PP decreases time spent to develop tasks when compared with solo programming.

H2. PP decreases the difficulty level to perform development tasks when compared with solo programming.

H3. PP increases correctness of developed programming tasks when compared with solo programming.

We defined the hypotheses H1 and H3 to guide our study in accordance with related work that investigate key-aspects of pair programming and agile methods [2][23]: impacts of development time spent and correctness of implemented features. In turn, considering that we run our experiments in academic context, we also are interested on investigating the difficulty faced by students to apply PP in information system development (hypothesis H2).

3.2 Participant Set

We obtained an initial participant set composed by 60 undergraduate or graduate students in Information Systems and related courses. Table 1 presents the total number of experiment participants. C1 is the first class (Experimental Software Engineering from UFMG), C2 is the second class (Software

Engineering from UIT), and C3 is the third class (Software Engineering from UFMG) where the experiment was applied.

Table 1. Participant Set

Participant Group	Class			Sum
	C1	C2	C3	
Pair Programming	16	12	12	40
Solo Programming	7	7	6	20
Total	23	19	18	60

3.3 Experiment Artifacts

We designed the following artifacts required to run the experiment.

- *Commitment Form*: this formal document: (i) describes the experiment, (ii) requests the use of collected data for analysis, and (iii) guarantees anonymous publication of obtained results.
- *Background Form*: we designed this six-question form to acquire participant data regarding previous knowledge in software systems development topics and techniques used in the experiment. Questions cover software development topics that participants may know, previous professional experience, and level of skill with respect to development techniques applied in our study.
- *Experiment Form:* this document describes the five programming tasks to be conducted in the one-hour experiment. It also includes fields to inform initial time, end time, and difficulty level for each development task. The difficulty level is an integer value between 1 (no difficulty) and 5 (unable to complete the task). This value aims to measure the overall difficulty faced by participants to complete an experiment task.
- *Feedback Form:* this form contains 9 questions to allow participants report their difficulty faced to: complete the experiment tasks, understand the source code provided for experiment execution, use the Java programming language, and use the Eclipse IDE. As in the Experiment Form, difficulty level is an integer value between 1 and 5.
- *Vending Machine information system:* this is a Java toy information system that participants used to develop tasks described in the Experiment Form.

The Vending Machine system simulates a soda vending machine. Users can insert coins with fixed value. Based on the inserted amount of coins, users can buy products available in the machine. Two classes totaling 261 lines of code (with JavaDoc comments) compose this system: Dispenser and Vending Machine. We also provided a support documentation in form of a class diagram. All artifacts used in the experiments, including the Commitment Form, the Experiment Form, and the source code of Vending Machine, are available for consultation in the research group website[3].

Experiment Tasks and Difficulty Levels. Because of time constraints and the context of application, we were not able to design experiment tasks that are excessively complex. Therefore, we conceived five simple experiment tasks to be conducted by participants, covering activities, such as item listing, value printing, arithmetic, and logic operations. The authors consider Tasks 1 and 3 the easiest experimental tasks, because they require only basic programming knowledge, such as printing of textual messages and data computed by the system. We required that participants conducted changes in these messages. In turn, Tasks 2, 4, and 5 are considered harder tasks to be solved because they required deeper knowledge regarding logical and arithmetic operations in addition to advanced object-oriented concepts.

3.4 Experiment Steps

We divided our experiments in the following four steps.

Step 1: Form Filling and Training. Before start with the experiment, participants filled the Commitment Form and the Background Form. Although none of the participants reported lack of knowledge about PP in the Background Form, we provided a brief description of the proposed experiment including basic concepts of pair programming.

Step 2: Experiment Presentation. Then, we described the experiment configuration for participants. We also instructed participants with respect to the required form filling. We presented the information system used to develop the experimental tasks and answered eventual questions about the experiment.

Step 3: Participant Randomization. For each class, we randomly divided participants in two groups: a control group to develop using solo programming and an experiment group to develop in pairs. Each experiment group was allocated to a different laboratory. We provided computers with the Eclipse IDE[4] installed and the source code of Vending Machine previously imported and opened.

Step 4: Experiment Execution. Participants (or pairs of participants) received an Experiment Form for filling during the experiment execution. Finally, participants filled the Feedback Form after concluding all tasks that they were able to complete in the limit time of experiment.

We conducted a pilot experiment with three volunteers to validate the feasibility of the proposed experiment. This pilot experiment supported us making decision regarding difficulty of developing the tasks, time constraints, form design, and applicability of the collected data in drawing conclusions. The pilot experiment execution also provided us feedback to refine the design of artifacts presented in Section 3.3. Results of the pilot experiment were discarded.

4. RESULTS AND DISCUSSION

This section presents and discusses the experiment results. Section 4.1 discusses the data set reduction. Section 4.2 presents an analysis with respect to the background of participants. Section 4.3 discusses results regarding the experiment execution, including the analysis of time spent, difficulty level, and correctness for each experiment task developed by participants using PP or solo programming. Section 4.4 analyzes the results of the experiment feedback provided by participants.

4.1 Data Set Reduction

In Section 3.2, we present our original set of participants for experiment execution. However, after running our experiment in the three classes, C1 to C3, we discarded five participants from

[3] http://goo.gl/qC7elq

[4] https://eclipse.org/downloads/

C2 due to two main reasons: problems during the experiment execution or unauthorized use of collected data for experimental analysis. Figure 1 shows the distribution of participants after the applied data set reduction.

Since we applied the experiment in classes, we rewarded participants with one point in course grade to motivate their participation. Moreover, the participation of students in the experiment was counted as presence in the course.

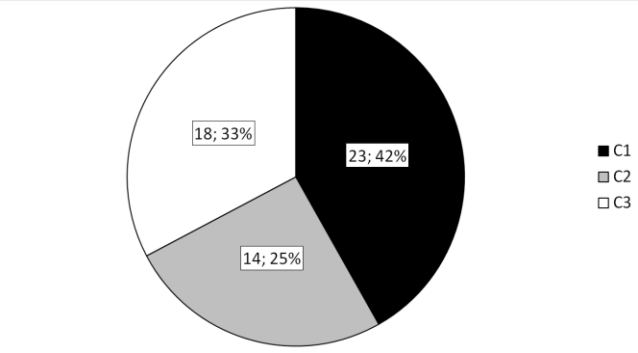

Figure 1. Distribution of Participants per Class

4.2 Background of Participants

In our data analysis, we first assessed the overall background of participants to identify a possible balance of participants in both PP and solo programming groups. For this purpose, we extracted data from the Background Form presented in Section 3.3.

Figure 2 presents the background of participants with respect to previous knowledge about relevant software systems development topics such as OO Programming, Software Modeling, and others. In Figure 2, we compare the percentage of participants from both groups (PP and Solo Programming) that informed they know about each topic. We observe both groups have almost the same reported knowledge for all studied topics, showing our randomization was able to distribute proportionally the participants through groups.

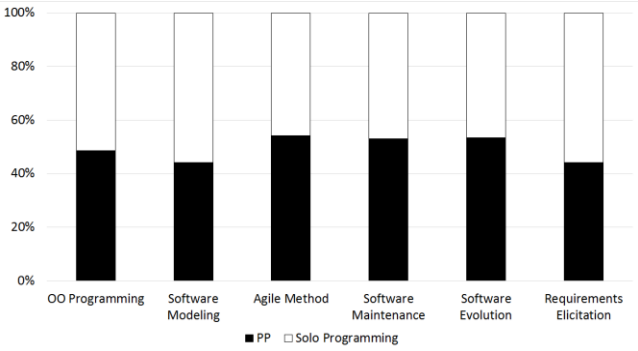

Figure 2. Background of Participants in Software Development Topics

Figure 3 illustrates the background of participants with respect to technical knowledge, covering professional experience and skills regarding UML Modeling, Java programming language, Eclipse IDE, and others. In Figure 3, we compare the percentage of participants that reported they have any experience with each topic. We considered any reported period of experience as professional experience of participants. We also counted any reported knowledge regarding the other technical topics as knowledge (except "none" option). Again, we observe a proportional knowledge distribution between PP and solo programming groups. Then, we conclude that our participation randomization provided a sufficient balance.

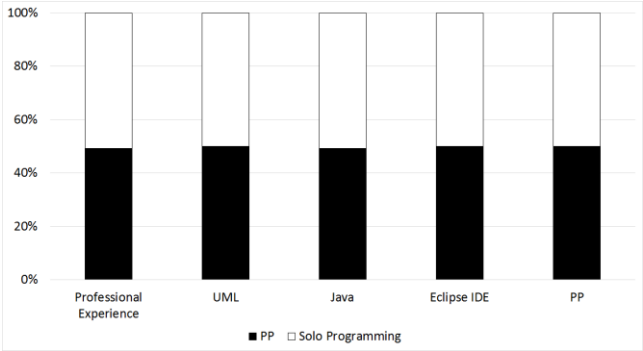

Figure 3. Technical Knowledge of Participants

4.3 Experiment Data

Through the analysis of collected data from the Experiment Form (see Section 3.3), we investigated each hypothesis proposed in Section 3.1. For each hypothesis, we divide our analysis in two steps: first, we present and discuss the obtained results for each experiment task by comparing both groups (PP and Solo Programming); second, we compare groups in terms of the distribution of results considering all the five tasks.

H1. PP decreases time spent to develop tasks when compared with solo programming.

Figure 4 illustrates the collected data with respect to time spent by participants to complete tasks, for each experiment group. We observe that PP developers spent significantly less time to complete the Tasks 2, 4, and 5 (the hardest tasks to complete, in the viewpoint of authors). Although Task 1 demanded less development time in PP than in solo programming group, PP participants reached lower rates of correctness for this task than other participants. In turn, considering Task 3, the PP group spent more time and less correctness than solo programmers. Note that Tasks 1 and 3 are considered the easiest tasks by authors.

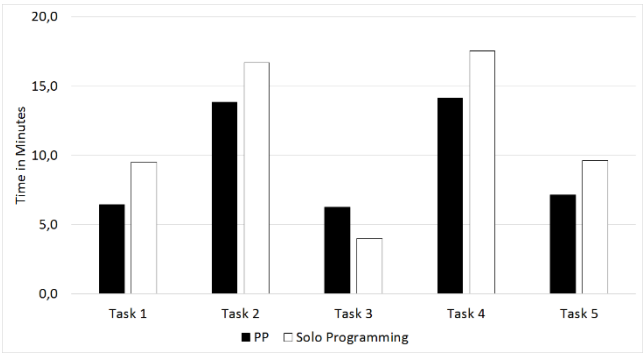

Figure 4. Time Spent by Groups per Task

Figure 5 presents the distribution of total time spent by participants to complete the experiment. In this figure, we have the distribution of time per group (PP and Solo Programming). Note that the three quartiles of the PP boxplot are above the respective quartiles in the Solo boxplot. Moreover, the median in time spent by pair is around 40 minutes, a lower value when compared with the median of 60 minutes for solo participants. Therefore, we conclude that PP supported a significant decrease of time spent to complete development tasks.

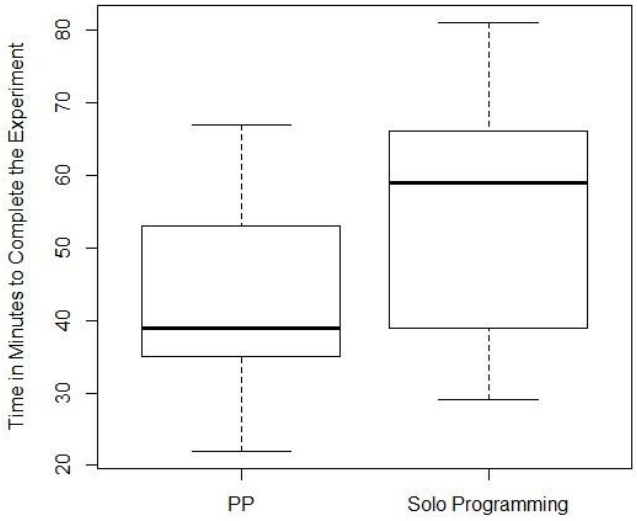

Figure 5. Time Spent to Complete the Tasks

H2. PP decreases the difficulty level to perform development tasks when compared with solo programming.

Figure 6 presents the results regarding difficulty levels faced by participants to complete tasks, per experiment group. We observe that PP developers reported significantly lower difficulty rates for 3 of 5 tasks, almost the same rate for 1 of 5 tasks, and higher rates for only 1 of 5 tasks. An overall analysis points that PP may decrease the difficulty to develop programming tasks when compared with solo programming. This observation confirms H2.

Considering that Tasks 2, 4, and 5 are the most difficult to complete in viewpoint of the study authors, we can conclude that PP may increase the developer comprehension of software requirements for a task. Therefore, PP may support knowledge and validation of a development task. These tasks were reported as the most difficult by paired participants when compared with solo programmers. However, in the easiest tasks from the viewpoint of authors, the difficulty level pointed by participants was approximately the same for both groups, PP and solo programming. This observation may point to an overestimation of difficulty level of simple tasks by PP participants.

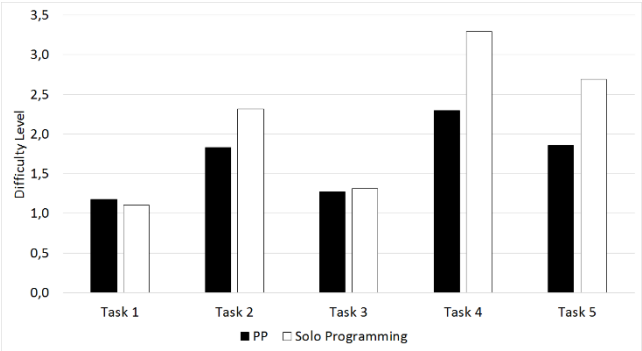

Figure 6. Difficulty Level of Groups per Task

Figure 7 presents the distribution of the mean difficulty level faced by participants to complete the experiment tasks. Note that the quartiles for paired participants are lower that the respective quartiles for solo participants. There is a significant decrease in difficulty in terms of median, for instance: pairs considered a mean of 1.5 in difficulty scale (from 1 to 5); in turn, solo participants have a difficulty around 2 in the same scale. Then, we conclude that PP may decrease difficulty faced in the development of programming tasks.

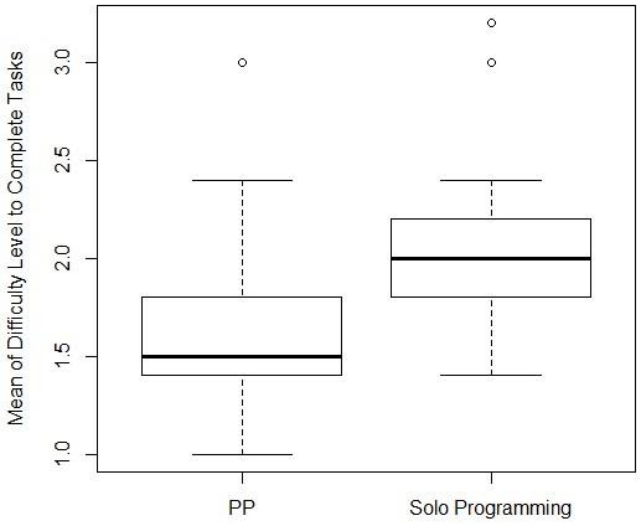

Figure 7. Mean of Difficulty Level to Complete Tasks

H3. PP increases correctness of developed programming tasks when compared with solo programming.

To investigate the hypothesis H3, we analyzed the source code sent by participants in the end of the experiment conduction. Figure 8 illustrates the obtained results. We observed slightly higher rates of correctness in tasks developed by PP participants when compared with solo programming for 3 of 5 tasks. However, this increase of correct development tasks is almost insignificant.

Figure 8 presents the percentage of participants that provided correct implementations for each experiment task. We observe that PP participants presented lower rates of correctness when compared with solo programmers with respect to the easiest tasks (Tasks 1 and 3). When considering the hardest tasks, PP showed a slightly increase of correctness in Task 5 and the same correctness rate for Tasks 2 and 4 when compared with solo programmers.

Considering only these results, we cannot affirm that H3 is correct. However, we believe that correctness of source code may be higher in PP groups than in solo programming groups. PP may be useful in comprehension of complex systems in terms of number of classes, attributes, methods, and packages. Furthermore, the development of a more realistic and robust system in terms of application domain may be benefited by the use of pair programming.

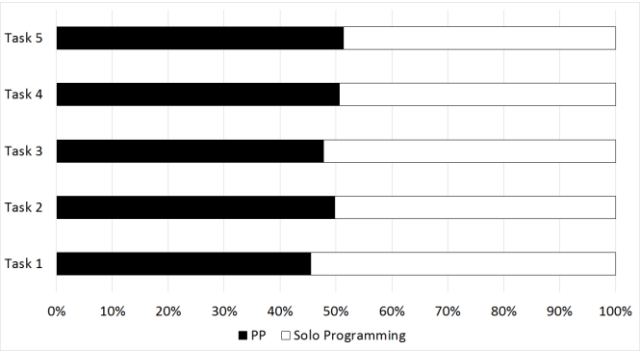

Figure 8. Correctness of Developed Tasks per Group

To support a different analysis, Figure 9 presents the distribution of correctness in answers from PP and Solo participants. In an opposite way when compared with Figures 5 and 7, this figure shows a slightly decrease of correctness in tasks performed by PP participants. In the authors' viewpoint, this situation occurred because of the simplicity of the experiment. If we consider that participants used a toy information system to develop tasks designed for undergraduate and graduate students, the simplicity of tasks may have contributed to excessive discussion regarding the solution of tasks and, consequently, implementation mistakes. We are based on the observations that PP may support increase of correctness provided by related studies that, in turn, analyzed the large-scale and industrial development context [2][23].

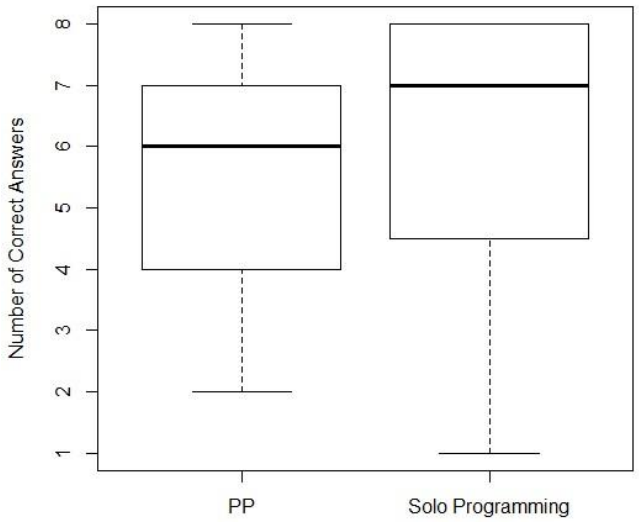

Figure 9. Distribution of Correct Answers

4.4 Feedback of Participants

Finally, we are also interested in participant feedback with respect to the use of PP to develop the proposed tasks. In this context, we provided a Feedback Form for filling in the end of the experiment execution. It consisted of nine questions to assess participant difficulty to comprehend the developed tasks, the source code, documentation, and others. Considering that most of the question in Feedback Form are related to the difficulty to comprehend and execute the experiment tasks (an aspect already analyzed in Section 4.3), we selected two questions for feedback analysis.

The first question for analysis is how difficult it was to use Java programming language in the experiment by participants. We aim to observe whether PP contributes to ease Java usage in class. Figure 10 presents the distribution of difficulty level reported by participants to use Java programming language in the experiment execution. We observe almost the same distribution of values for both PP and Solo participants. However, we can notice that the median of difficulty for PP participants is lower than for solo programmers. Although these results are a subjective viewpoint of participants, we point that PP did not significantly affect the use of Java in the experiment.

The second question we analyze is about how difficult was to use the Eclipse IDE in the experiment by participants. Considering that participants include undergraduate students that may not know tools such as Eclipse, we aim to observe the impact of PP to ease the usage of this tool. Figure 11 presents the distribution of difficulty faced by participants to use Eclipse IDE per group. Again, we observe no significant difference between difficulty level faced by PP and Solo groups to use the Eclipse IDE.

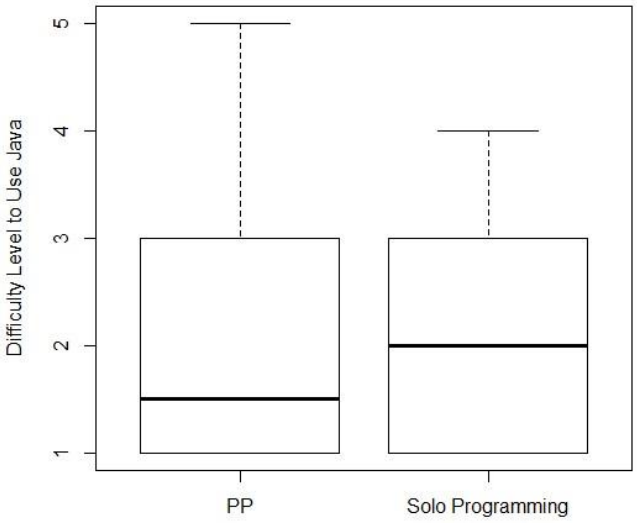

Figure 10. Distribution of Difficulty to Use Java Language

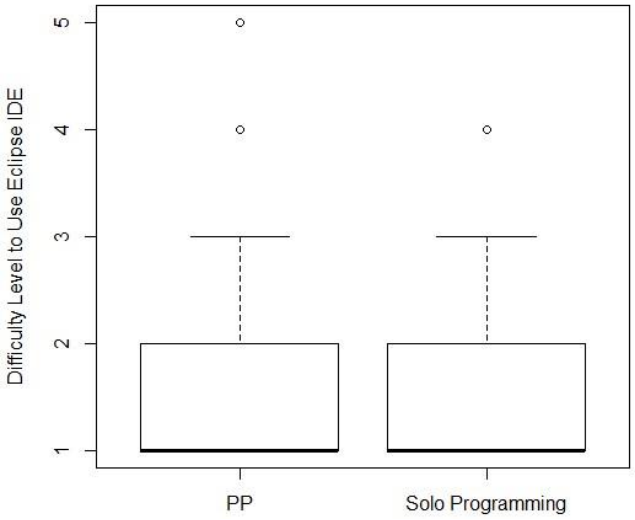

Figure 11. Distribution of Difficulty to Use Eclipse IDE

In the authors' viewpoint, we are not able to identify significant advantages of using PP to support ease of learning and using development technologies (in case, Java and Eclipse IDE) because of the simplicity of our experiment. In the experiment execution, we did not require from participants the usage of advanced features of Java, for instance. However, we expect that in the development of large-scale and complex information systems PP may be more helpful, as indicated by related work [2][23].

5. RELATED WORK

We were able to find few studies that investigate specifically the application of pair programming in the industrial context [2][20][23]. For instance, Vanhanen and Lassenius [23] studied the impacts of using pair programming in comparison with single programming in large-scale software development. They conducted a survey with 28 experienced developers to investigate the relationship of pair programming with software quality, development effort, learning, and other aspects of software

development. They conclude pair programming may support learning and team collaboration, for instance. They also observed a slightly increase of software quality when programming in pairs.

Arishholm et al. [2] conducted an extensive experiment with 295 developers with different professional experiences, from 29 consultancy companies in three countries. This experiment aimed to assess the use of pair programming in development tasks. The authors observed a significantly positive impact of using pair programming only in case of complex and large software system development, including an increase of correctness without decreasing the time spent to develop programming tasks. It was also observed a higher support of PP in development activities with respect to juniors when compared with experienced developers. Finally, Prechelt et al. [20] conduct an experiment to assess advantages of using pair programming in academic context. The authors aim to provide guidelines to support the execution of empirical studies in industry, such as PP assessment, that are generalizable to other contexts such as academic context.

Previous work investigate the use of pair programming in the academic context [12][15][16][24]. McDowell et al. [16] investigate the impact of pair programming on the performance of undergraduate students in class. About 600 students composed the participant set. The participants were divided in groups to assess the use of pair programming when compared with single programming. As a result, developers in pairs produced software with higher quality and obtained higher grades in the course when compared with other participants. They also discuss that pair programming may be useful in class. McChesney [15] presents another study. In this work, three years of experiments using pair programming are conducted to assess contributions of PP on computer science education. The author observed many advantages of using PP in academy including improvement of student performance and pairwise communication.

Williams et al. [24] conducted experiments with over 1200 undergraduate students in order to assess the efficacy of pair programming in the academic context. These experiments were applied in two educational institutes in US. They concluded that pair programming may support the student formation, improving their programming skills when compared with the use of individual programming. Finally, Hannay et al. [12] conducted a meta-analysis regarding the experiments with pair programming both in industry and academy. This study covered experiments that compare the use of pair in development with single programming. They conclude that, in general, pair programming may support software quality, although with significant impact on time spent and effort to developers. However, they point common bias in studies with respect to the comparison of pair programming with other development practices.

6. THREATS TO VALIDITY

We designed and conducted carefully the three experiments described in Section 3. For instance, we delimited our experiment scope prior to the execution of the experiments, defined our hypotheses, and how to assess them, after study and based on previous studies. However, some threats to validity may affect our research findings. Following, we discuss each of the four types of threats, with respective treatments, listed by Wolin et al. [25]: internal, conclusion, construct, and external validity.

Construct Validity. We designed our experiments to be applied in different education institutions without significant adaptations, in order to prevent the decharacterization of the experiment. We conducted pilot experiments with volunteers to define development tasks that could be adequately performed by students considering aspects such as experiment time, difficulty level, and the use of Java language in academic context. To provide impartiality of participants with respect to our research questions and experiment hypotheses, we omitted this information to all participants. Therefore, we expected an experiment execution with minimization of biases. Finally, we proposed an oracle for each task, to base the assessment of correctness to each developed task.

Internal Validity. Participants collected data regarding time spent and difficulty level for each developed task through the Experiment Form. However, this procedure may be affected by the comprehension of participants with respect to each proposed task. To minimize this problem, we provided a brief tutorial for form filling, as well as code documentation, and supported participants during the experiments in case of difficulty to understand tasks. We also helped participants in case of problems with Java language, but it was not recurrent mainly because we provided Internet access to support. Moreover, participants may have been unmotivated to complete the experiment. In this context, we rewarded participants with scores in course grade. Our findings may also be affected by the unbalance between participant groups. To minimize this problem, we randomly allocated participants in groups with approximately same size. We discuss the obtained balance in Section 4.

Conclusion Validity. We conducted a careful data analysis to draw conclusions regarding the applied experiments, to minimize problems with respect to data interpretation. We also chose carefully descriptive analysis techniques to present results appropriately. We based our selection of data to be collected in related work, to ensure that such data would be useful in drawing conclusions.

External Validity. Some factor may prevent the generalization of our research findings. For instance, the 55 participants consisted of undergraduate and graduate students from two educational Brazilian institutions. They may not represent properly all Brazilian students, considering their amount, background, cultural aspects, professional experience, and others. However, the available participants for experiment were randomly distributed in groups to minimize this problem and increase the representativeness of the groups. Furthermore, we restricted our experiment execution to one hour, because the experiments were conducted in class. This constraint may affect our findings, since participants may be uncomfortable to develop under time restrictions. However, our experiments were placed in laboratories equipped with sufficient computers and all required tools (Eclipse IDE, for instance) to appropriate experiment execution.

7. CONCLUSION

In this study, we investigated the use of PP in academic context, with undergraduate and graduate students from two educational Brazilian institutions. For this purpose, we run three one-hour experiments composed by five development tasks to be conducted using Java and Eclipse IDE. We compare performance of students using PP and solo programming with respect to: difficulty level faced by participants to complete tasks, time spent, and correctness of implementations provided by participants for tasks.

In general, we observed a positive impact of PP on development regarding time spent and difficulty to complete tasks. Although paired students presented significantly lower difficulty levels and time spent to develop almost all the proposed development tasks

when compared with solo programmers, we were not able to observe an increase of correctness in source code implemented by pairs. Therefore, we conclude PP may support software development performance and product delivery. However, it is necessary another study to assess accurately the possible increase of software quality when using PP as a development practice.

We also observe significant indications that PP may be more efficient when compared with solo programming, considering complex and large software systems in terms of lines of code, number of entities, complexity of application domain and other software aspects. Therefore, PP may be useful in requirements comprehension and software verification, decreasing costs with software maintenance, for instance. However, for simple systems, PP may not be more efficient than solo programming, requiring unnecessary time to develop tasks, for instance.

As future work, we suggest the application of this experiment in other institutions to cover more participants from different estates and contexts. A larger set of participants, with more diversified background and professional experiences, may be interesting to increase the generalization of our findings. We also suggest a more robust statistical analysis, to support drawing conclusion regarding the application of PP in academic environment.

8. ACKNOWLEDGMENTS

This work was partially supported by CAPES, CNPq (grant 485907/2013-5), and FAPEMIG (grant PPM-00382-14).

9. REFERENCES

[1] Ambler, S. and Lines, M. 2012. *Disciplined Agile Delivery: A Practitioner's Guide to Agile Software Delivery in the Enterprise*. IBM Press.

[2] Arisholm, E., Gallis, H., Dyba, T., and Sjoberg, D. 2007. Evaluating Pair Programming with Respect to System Complexity and Programmer Expertise. *Transactions on Software Engineering (TSE)* 33, 2, 65-86.

[3] Beck, K. 2000. *Extreme Programming Explained: Embrace Change*. Addison-Wesley Professional.

[4] Begel, A. and Nagappan, N. 2007. Usage and Perceptions of Agile Software Development in an Industrial Context: An Exploratory Study. In *Proceedings of the 1st International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 255-264.

[5] Begel, A. and Nagappan, N. 2008. Pair Programming: What's in it for me?. In *Proceedings of the 2nd International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 120-128.

[6] Bhat, T. and Nagappan, N. 2006. Evaluating the Efficacy of Test-Driven Development: Industrial Case Studies. In *Proceedings of the 5th International Symposium on Empirical Software Engineering (ISESE)*, 356-363.

[7] Cockburn, A. 2004. *Crystal Clear: A Human-Powered Methodology for Small Teams*. Pearson Education.

[8] Cockburn, A. and Williams, L. 2000. The Costs and Benefits of Pair Programming. *Extreme Programming Examined*, 223-247.

[9] Cohen, D., Lindvall, M., and Costa, P. 2004. An Introduction to Agile Methods. *Advances in Computers* 62, 1-66.

[10] Dyba, T., Arisholm, E., Sjoberg, D. I., Hannay, J. E., and Shull, F. 2007. Are Two Heads Better than One? On the Effectiveness of Pair Programming. *Software* 24, 6, 12-15.

[11] Fowler, M. and Highsmith, J. 2001. The Agile Manifesto. *Software Development* 9, 8, 28-35.

[12] Hannay, J., Dyba, T., Arisholm, E., and Sjoberg, D. 2009. The Effectiveness of Pair Programming: A Meta-Analysis. *Information and Software Technology* 51, 7, 1110-1122.

[13] Janzen, D. and Saiedian, H. 2008. Does Test-Driven Development Really Improve Software Design Quality? *Software* 25, 2, 77-84.

[14] Kavitha, R. and Ahmed, M. 2015. Knowledge Sharing through Pair Programming in Learning Environments: An Empirical Study. *Education and Information Technologies* 20, 2, 319-333.

[15] McChesney, I. 2016. Three Years of Student Pair Programming: Action Research Insights and Outcomes. In *Proceedings of the 47th Technical Symposium on Computer Science Education (SIGCSE)*, 84-89.

[16] McDowell, C., Werner, L., Bullock, H., and Fernald, J. 2002. The Effects of Pair-Programming on Performance in an Introductory Programming Course. In *Special Interest Group on Computer Science Education Bulletin (SIGCSE Bulletin)* 34, 38-42.

[17] Nagappan, N., Maximilien, E., Bhat, T., and Williams, L. 2008. Realizing Quality Improvement through Test Driven Development: Results and Experiences of Four Industrial Teams. *Empirical Software Engineering (ESE)*, 13, 3, 289-302.

[18] Paulk, M. 2001. Extreme Programming from a CMM Perspective. *Software*, 18, 6, 19-26.

[19] Plonka, L., Sharp, H., Van der Linden, J., and Dittrich, Y. 2015. Knowledge Transfer in Pair Programming: An In-Depth Analysis. *International Journal of Human-Computer Studies*, 73, 66-78.

[20] Prechelt, L., Zieris, F., and Schmeisky, H. 2015. Difficulty Factors of Obtaining Access for Empirical Studies in Industry. In *Proceedings of the 3rd International Workshop on Conducting Empirical Studies in Industry (CESI)*, 19-25.

[21] Schwaber, K. 1997. Scrum Development Process. *Business Object Design and Implementation*, 117-134.

[22] Schwaber, K. 2004. *Agile Project Management with Scrum*. Microsoft Press.

[23] Vanhanen, J. and Lassenius, C. 2007. Perceived Effects of Pair Programming in an Industrial Context. In *Proceedings of the 33rd Conference on Software Engineering and Advanced Applications (EUROMICRO)*, 211-218.

[24] Williams, L., McDowell, C., Nagappan, N., Fernald, J., and Werner, L. 2003. Building Pair Programming Knowledge through a Family of Experiments. In *Proceedings of the 2nd International Symposium on Empirical Software Engineering (ISESE)*, 143-152.

[25] Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., & Wesslén, A. 2012. *Experimentation in Software Engineering*. Springer Science & Business Media.