Pair Programming and Software Defects—A Large, Industrial Case Study

Enrico di Bella, Ilenia Fronza, Nattakarn Phaphoom, Alberto Sillitti, *Member*, *IEEE*, Giancarlo Succi, *Member*, *IEEE*, and Jelena Vlasenko

Abstract—In the last decade, there has been increasing interest in pair programming (PP). However, despite the existing work, there is still a lack of substantial evidence of the effects of PP in industrial environments. To address this issue, we have analyzed the work of a team of 17 industrial developers for 14 months. The team is part of the IT department of a large Italian manufacturing company; it adopts a customized version of extreme programming (XP). We have investigated the effects of PP on software quality in five different scenarios. The results show that PP appears to provide a perceivable but small effect on the reduction of defects in these settings.

Index Terms—Pair programming, software defects, case study

1 Introduction

Pair programming (PP) is a practice when two developers work together on the same task using one computer [5], [83], [84]. A pair is composed of a "driver" and a "navigator." While the driver writes a program code, the navigator does more strategic tasks, e.g., reviewing the code, searching for tactical errors, thinking of the overall architecture, finding better alternatives, and brainstorming [5]. PP has gained a lot of attention in the last decade as one of the extreme programming (XP) practices [5]. However, this technique was introduced years before becoming a part of XP [84].

It has been alleged that PP may improve software development under several perspectives. A large number of empirical studies have been conducted and their results have been used to support these allegations. The apparent benefits of PP include:

1. reducing a defect rate [81], [78], [60], [46], [77], [77], [6], [67], [68],
2. improving design [10], [11],
3. improving the structure of the code [87],
4. increasing productivity [32], [41], [39], [49], [81],
5. shortening time-to-market [61], [60], [57], [81], [13], [20], [30],
6. enhancing knowledge transfer and team communication [12], [77], [6], [14], [78], [13], [84],
7. increasing job satisfaction [6], [73], and
8. facilitating the integration of newcomers to the team and reducing training costs [23], [24], [82].

- *E. Di Bella is with the Department of Economics and Quantitative Methods, University of Genova, Via Vivaldi 5, 16126 Genova, Italy. E-mail: edibella@economia.unige.it.*
- *I. Fronza, N. Phaphoom, A. Sillitti, G. Succi, and J. Vlasenko are with the Center for Applied Software Engineering (CASE), Free University of Bozen-Bolzano, Piazza Domenicani 3, 39100 Bolzano, Italy. E-mail: {ilenia.fronza, nattakarn.phaphoom, alberto.sillitti, giancarlo.succi}@unibz.it, jelena.vlasenko@stud-inf.unibz.it.*

Manuscript received 2 Sept. 2011; revised 25 Apr. 2012; accepted 8 Oct. 2012; published online 15 Oct. 2012.
Recommended for acceptance by L. Williams.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number TSE-2011-09-0252.
Digital Object Identifier no. 10.1109/TSE.2012.68.

However, there are also a number of studies that do not confirm these results, especially regarding productivity and cost-efficiency. In an experiment conducted by Nawrocki and Wojciechowski, no positive effect of PP on development time is found [55]. In a case study conducted by Hulkko and Abrahamsson, pair productivity and the quality of the resulting software product vary across projects [33]. A large experiment conducted by Arisholm et al. shows that PP neither reduces the time required to correctly perform change tasks nor increases the percentage of correct solutions [2]. Its effectiveness depends on the following confounding factors: 1) Correctness increases when junior pairs work on a complex system, and 2) duration decreases when nonjunior pairs work on a simple system. Results of the survey conducted by Begel and Nagappan at Microsoft indicate that there is a high level of skepticism over pair efficiency inside the organization [6].

Replicability of the experiments and generalizability of the findings are well-known challenges involved in designing PP research [2], [30], [10], [40]. So far, variations of experimental designs have been reported and results gained from each of these experiments differ considerably. This is mainly due to situational variables and confounding factors that are difficult to control. Lui et al. summarize the major causes of inconsistency of the results across PP studies [40]; these factors include the pairing process, pair configuration, programming tasks, task complexity with regard to skill and experience of subjects, the order in which different tasks are given, hidden variables between programming tasks, and the effect of pair jelling. Difficulties in controlling such factors limit the ability of researchers to replicate previous experiments and to determine the situations in which PP is effective.

Moreover, it is difficult to generalize the results of PP studies beyond their experimental settings to the overall software industry. First of all, most of the experiments have been conducted in educational environments using students as experimental subjects [81], [55], [3], [69], [82], [47], [54], [22], [49], [75], [1], [7], [28], [50], [78], [86], [10], [43], [45], [51],

TABLE 1
Inclusion and Exclusion Criteria for the Literature Review

Inclusion criteria	Exclusion criteria
1. The study reports performance of PP in terms of quality of results using clearly defined quality metrics. 2. The study is a case study, a survey, an experiment, or a literature review. 3. The study was published after January 1998.	Studies conducted with students in an introductory computer science (CS) course are excluded.

[44], [52], [61], [8], [19], [67], [68], [9], [21], [48], [56], [63], [64], [62], [88]. The educational environment might not represent precisely the real working environment in a company where a team of developers works collaboratively on given tasks. As argued, e.g., by Höst et al., it is not uncommon to use students as subjects when conducting empirical investigations [94]. The students are readily available, often willing to participate, and require little or no compensation. However, one cannot use students blindly and then generalize the findings to a larger population of software engineering professionals [90], [94]. Even though there are some preliminary indications that students can be used for certain tasks instead of professionals under certain conditions [94], it is still unclear how well results from student-based experiments generalize to professional software engineers [93]. Moreover, the tasks that the subjects performed in the available studies are often the implementation of small and isolated programs or the introduction of small changes into existing programs [57], [55], [22], [18], [19]. Such tasks are not representative for the typical tasks in companies.

So far there has been no investigation of the specific effect that PP has on coding, for instance when developers create/correct a method or a class. Empirical studies in this area focus on the immediate effect that PP has on the quality of the code based on a limited study time frame. As a consequence, the long-term effect of PP is not well understood. Moreover, the existing studies are more generic, typically comparing tasks done in pairs and alone under some quality metrics [68], [19], [39], [67], [11], [10], [1], [86], [78], [49], [41], [3], [55], [81], [57].

It is, however, essential to understand the effects of PP as it is used whenever developers find it appropriate and beneficial, rather than being forced to use it. This could be done by analyzing the long-term effect of PP in an industrial setting where the task itself enacts the need of PP. The contribution of this paper is thus to enhance the understanding of long-term PP effects in this setting. Contrary to past PP studies which try to understand how pairs can implement high-quality code in less time, we have analyzed the relationship between the amount of PP usage and the defects in the source code in five situations to answer the following main research questions: 1) Does PP help to reduce the introduction of new defects when it is used for defect corrections and implementations of user stories? and 2) does the enhanced knowledge of the code by long-term regular usage of PP help to reduce defects? The analysis is based on 14-month data collected from an XP team of a large Italian manufacturing company that prefers to remain anonymous.

As mentioned, the ultimate goal of our analysis is to provide evidence as to whether the code contains fewer defects when developers do PP. To answer this question, we measure the PP usage on each unit of the source code (a method in classes) and the defect density (DD) of the method. Then, we use statistical models to visualize the trend of both variables. Finally, we use statistical tests to verify if there is a significant difference between the group of methods developed/modified during PP and the group of methods developed/modified during solo programming.

This paper is organized as follows: Section 2 summarizes the related work. Section 3 discusses the research design and hypotheses. The analysis methods are explained in Section 4 and the results of the analysis are presented in Sections 5 and 6. The threads to validity are discussed in Section 7, and finally, conclusions are presented in Section 8.

2 Related Work

The goal of the literature review is to identify under which circumstances PP can effectively improve software quality in industrial environment. Following the guidelines proposed by Kitchenham [37], we consider the following research questions:

Under which circumstances does PP increase software quality in industrial projects?

- RQ1: Under which circumstances has the impact of PP on code quality been investigated?
- RQ2: What are the main findings of the impact of PP on code quality?

We searched for existing works in the ACM Digital Library, IEEE Xplore, ScienceDirect, and Springer with a search string "pair programming" on a title, a keyword, and an abstract. Then, we applied inclusion and exclusion criteria to the identified studies. These criteria are explained in Table 1. As the third inclusion criterion indicates, we limited the search to work published after January 1998 because the first formal experiment on PP conducted by Nosek [57] was published that year [81]. The last search was done in June 2011.

In the exclusion criteria, we omit studies conducted with students in introductory computer science (CS) courses. This is due to the fact that how to generalize the results of such studies to industrial settings is very controversial [79]. For instance, the level of programming expertise and the complexity of performed tasks can influence the performance of PP. Table 2 summarizes the differences of the work done by students of CS introductory courses and by professionals in the industry under the perspective of three

TABLE 2
Analysis of the Differences between Code Development in Introductory CS Courses and in Professional Settings with Respect to Three Situational Variables—Subject, Task, and Environment

Situational variables	Students in CS introductory courses	Professional developers in companies
Subject	• No or limited programming experience • No PP experience • No or limited knowledge in task domain	• Average to high programming experience • Mostly limited PP experience • Average to high knowledge in task domain
Task	Simple, individual, and independent development tasks aiming to teach students a new programming language.	New development, feature enhancements, defect corrections, and other maintenance tasks. The tasks are implemented on a large and complex system with major dependencies between modules. In such cases minimizing impact and good design are greatly considered.
Environment	Students usually follow informal development processes and practices.	Developers often follow formal processes and are equipped with standard software.

situational variables: the experiences of the subject performing the task, the features of the tasks, and the overall development environment.

The search criteria resulted in identifying 30 primary studies (Table 3). They involve different subjects, production settings, experimental designs, and geographical locations:

- Seventeen studies have as subjects experienced students (S), 10 have professional developers (P), and three mix both students and professionals (M).
- Twenty-four studies are an experiment, four are a case study, and two are a survey.
- Twenty-two studies are related to programming, five to designing, one to testing, and two are surveys.
- Fifteen studies are from Europe (EU), eight are from North America (North AM), four are from Asia, one is from Australia (AU), one is from Central America (Central AM), and one is an international survey (Intl).

By considering different configurations on *subjects and teams*, these studies reveal two abstraction levels in which PP studies were conducted. Studies that analyze PP effects on the team level generally compare the effectiveness of PP teams against solo teams. Studies that analyze the effectiveness of the practice itself compare the work of pairs against that of solos. We differentiate these two settings by using the term (n PP teams, n teams of solos) to indicate the team-level study, and using (n pair, n solos) to indicate the practice-level study.

As indicated by *scope*, we classified the main focus of the studies into four categories, i.e., designing, programming ($design + code + test$), testing, and survey. This classification is based on the activities the subjects performed to accomplish the task. The scope corresponds to the *artifacts being evaluated* in the end of the case study or the experiment. The studies related to designing are concerned with design strategies, rationales, and design product maintenance in which the quality of design artifacts such as use cases, class diagrams, and work flows are evaluated as an end result. The studies related to programming are concerned with the development or changes of code artifacts, which generally require the subjects to undergo a process of design, coding, and testing to achieve a desired level of product quality. The quality of code is then evaluated through different measures such as defect counts, DD, or code metrics. The study related to testing is concerned with test artifacts, such as test scripts, in which the ability of detecting defects is evaluated.

A study by Hannay and Jorgensen suggested a set of situational variables that should be considered when generalizing the results of the experiments [29]. These variables include subjects, treatments, outcomes, settings, tasks, and material. We classify the included studies according to such variables to answer RQ1.

Fig. 1 summarizes subjects and tasks involved in the 30 selected studies. Most of the studies dealing with experienced (i.e., advanced undergraduate and graduate) students are related to the coding task (14 out of 17) and three to the design task. The studies dealing with professionals are related to the design task (2), the coding task (5), testing (1), and survey (2). Three studies dealing with both students and professionals are related to the coding task. The studies related to coding were further classified to new development and change as they fundamentally require a different skill set. New developments refers to the work that was developed from scratch, while changes refers to enhancements or defect corrections on an existing system. Altogether, only four studies related to coding deal with changes in an existing system.

Furthermore, the experimental studies are classified according to the treatment given to the experimental and control groups (Table 4). Among 24 experiments, five are related to design, 18 to coding, and only one is related to testing. Five experiments compare the effectiveness of pair and solo design. Among experiments related to coding, 10 compare PP with solo programming in a colocated environment. Four experiments verify the effectiveness of distributed pairs and four experiments compare PP with other SE techniques (e.g., code inspection and personal software practice). Only one experiment compares the effectiveness of writing test scripts between solos and pairs.

The effects of PP on the quality of the products are explored using various metrics. Fig. 2 outlines the quality metrics used in the 30 selected studies. Out of five studies on design, three use a score that is evaluated by an expert or quality checklists to measure the correctness of a solution, one study uses various measurable attributes of the ISO 9126 definition of quality, and one uses the number of assignment submissions with incorrect answers. Studies related to

TABLE 3
Studies Selected for the Analysis

Id	Study	Geographical location	Scope	Evaluated artifact	Study type	Subject	Number of subjects
1	[Nosek, 98]	North AM	Programming (design-code-test)	Code	Experiment	P	15 (5 pairs , 5 solos)
2	[Williams *et al.*, 00]	North AM	Programming (design-code-test)	Code	Experiment	S	41 (14 pairs, 13 solos)
3	[Nawrocki and Wojciechowski, 01]	EU	Programming (design-code-test)	Code	Experiment	S	21 (6 PSP solos, 5 XP solos, 5 XP pairs)
4	[Baheti *et al.*, 02]	North AM	Programming (design-code-test)	Code	Experiment	S	132 (9 co-located teams of solos, 16 co-located teams of pairs, 8 distributed teams of solos, 5 distributed teams of pairs)
5	[Stotts et al, 03]	North AM	Programming (design-code-test)	Code	Case study	S	8 (2 distributed pairs, 2 distributed teams)
6	[Favela *et al.*, 04]	Central AM	Programming (design-code-test)	Code	Experiment	S	12 (2 co-located pairs, 2 distributed pairs, 2 distributed pairs)
7	[Al-Kilidar *et al.*, 05]	AU	Designing	Design	Experiment	S	150 (29 pairs, 58 solos, 34 solos in a controlled group)
8	[Bellini *et al.*, 05]	EU	Designing	Design	Experiment	S	44 (5 pairs with technology background, 5 pairs with management background, 4 mixed pairs, 16 solos)
9	[Hulkko and Abrahamsson, 05]	EU	Programming (design-code-test)	Code	Case study	M	4 teams of 4-6 developers
10	[Vanhanen and Lassenius, 05]	EU	Programming (design-code-test)	Code	Experiment	S	20 (3 PP teams, 2 solo teams)
11	[Xu and Chen, 05]	North AM	Programming (design-code-test)	Code	Experiment	S	6 (2 pairs, 2 solos)
12	[Canfora *et al.*, 06]	EU	Designing	Design	Experiment	S	60
13	[Madeyski, 06]	EU	Programming (design-code-test)	Code	Experiment	S	188 (28 solos, 28 solos TDD, 31 pairs, 35 TDD pairs)
14	[Müller, 06]	EU	Programming (design-code-test)	Code	Experiment	S	18 (1 PP team, and 1 team of solos)
15	[Phongpaibul and Boehm, 06]	Asia	Programming (design-code-test)	Code	Experiment	M	95 students (7 PP teams, 7 inspection teams), 8 professionals (1 team of pairs, 1 inspection team)
16	[Arisholm *et al.*, 07; Hannay *et al.*, 10]	EU	Programming (design-code-test)	Code	Experiment	P	295
17	[Canfora *et al.*, 07]	EU	Designing	Design	Experiment	P	18 (6 pairs, 6 solos)
18	[Domino *et al.*, 07]	North AM	Programming (design-code-test)	Code	Experiment	M	216
19	[Madeyski, 07]	EU	Testing	Test	Experiment	S	98 (35 pairs, 28 solos)
20	[Müller, 04; Müller, 05; Müller, 07]	EU	Programming (design-code-test)	Code	Experiment	S	38
21	[Phongpaibul and Boehm, 07]	North AM	Programming (design-code-test)	Code	Experiment	S	56 (5 PP teams, 4 inspection teams)
22	[Vanhanen and Korpi, 07]	EU	Programming (design-code-test)	Code	Case study	P	Team of 4 developers
23	[Vanhanen and Lassenius, 07]	EU	Survey	n/a	Survey	P	27
24	[Begel and Nagappan, 08]	Intl	Survey	n/a	Survey	P	106
25	[Duque and Bravo, 08]	EU	Programming (design-code-test)	Code	Experiment	S	51 (17 pairs, 17 solos)
26	[Lui *et al.*, 08; Lui and Chan, 03]	Asia	Designing	Design	Experiment	P	15 (5 pairs, 5 solos)
27	[Sison, 08]	Asia	Programming (design-code-test)	Code	Experiment	S	24 (6 PP teams, 6 teams of solos)
28	[Sison *et al.*, 09]	Asia	Programming (design-code-test)	Code	Experiment	S	48 (16 pairs, 16 solos)
29	[Keeling, 10]	North AM	Programming (design-code-test)	Code	Experiment	S	4 (1 PP team, 1 team of solos)
30	[Phaphoom *et al.*, 11]	EU	Programming (design-code-test)	Code	Case study	P	17

coding (22 studies) adopt various quality measures. Seven studies use a score evaluated by experts or teaching assistants; another seven studies use the number of defects or failures found in code; five use DD; three studies use code metrics, e.g., comment ratio, proportion of bad methods evaluated by the method's length and complexity, and density of coding standard deviations. One study uses several code metrics to measure dependence between packages. Two empirical surveys use voting scores to measure perceived effects of PP. Only one study related to testing uses test coverage measure and mutation score to investigate the effectiveness of pair testing.

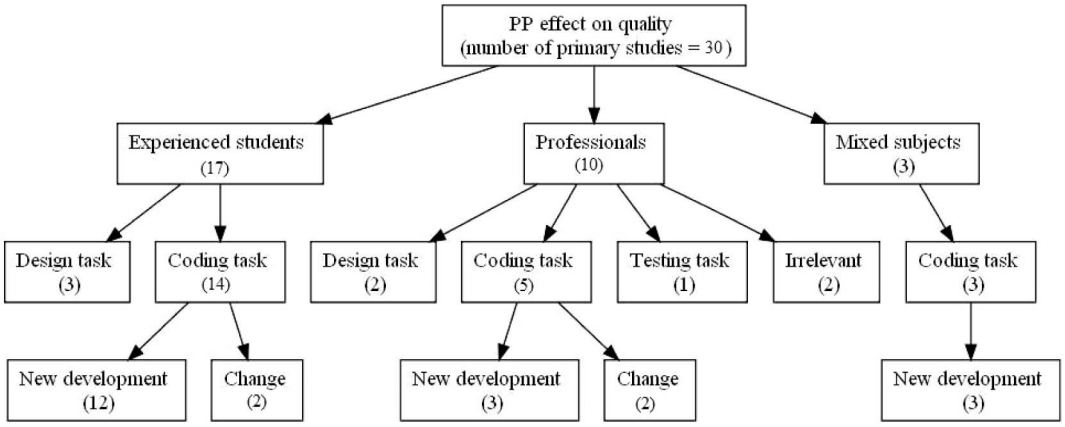


Fig. 1. Subjects and tasks involved in the selected studies.

TABLE 4
Distribution of Experimental Studies According to the Treatment Given to Experimental and Control Groups

Evaluated artifacts	Comparison between treatments			
	Co-located pair vs. Solo	Distributed pair vs. Co-located pair / Solo	Co-located pair vs. Inspection	Co-located pair vs. other SE practices
Design	5	n/a	n/a	n/a
Code	10	4	3	1
Test	1	n/a	n/a	n/a

Based on the results of RQ1, the design of experiments and case studies on PP is divergent. Different combinations of situational variables, especially the measurements of outcomes (Fig. 2), the treatment given to experimental and controlled groups (Table 3), and the features of the tasks (Fig. 1) make it questionable to combine the results for all these studies.

Meta-analysis provides a possibility of combining the results from many primary studies into a single result. Combining small studies to get a large sample can increase the possibility for statistical tests to detect small effects. However, combined studies should be performed in a similar way or the difference in their experimental design should be ruled out. The studies by Arisholm et al. [2] and Hannay et al. [30] suggest several confounding factors that hinder a statistically sound determination of PP effectiveness: developer expertise, task complexity, and country. However, the classification themes of such variables are not standardized across the selected studies. Taking all these issues into consideration, we have decided to organize results into the different cases.

To answer RQ2, the results of the selected studies have been summarized according to the scope of the performed tasks, i.e., design, coding, and testing. The studies related to coding are presented in four settings, including colocated PP, distributed PP, PP with other practices, and industrial perception. In short, the results are organized as follows:

- Pair design (Table 5).
- Colocated PP (Table 6).
- Distributed PP (Table 7).
- PP with other practices (Table 8).
- Perception of PP from industrial practitioners (Table 9).
- Pair testing (Table 10).

The situational variables capture an experimental design and give an insight to the result and analysis. We have systematically summarized the results of each study by presenting the treatment, the employed quality metrics, the performed task, the effective cases, and the outcomes. The "effective case" is the ratio of the number of "cases" when PP groups outperform the compared groups over the total number of tested cases. We prefer to use the term "case" instead of "hypothesis" to generalize the situation in which the experimental and control groups are compared. The notion of hypothesis requires a statistical test to be performed to confirm that the differences in the observations are not coincidental [85]. However, the statistical tests were not performed in some of the included studies [69], [22], [33], [78], [86], [19] and some studies do not compare PP and solo

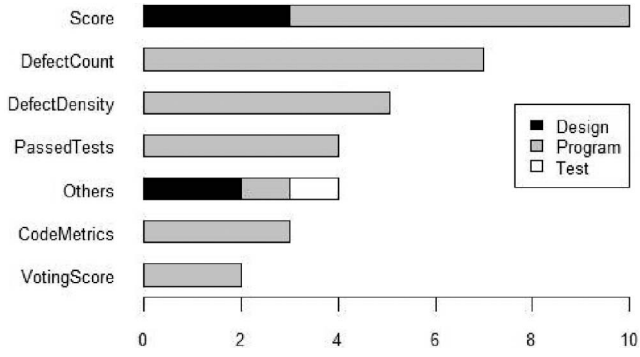


Fig. 2. Distribution of the quality metrics used in the 30 selected studies.

TABLE 5
Summary of Results for Pair Design

Study	Treatment (quality metric)	Task description	Effective cases	Outcomes and comments
[Lui *et al.*, 08], [Lui and Chan, 03]	Pair-solo (a number of submissions with incorrect solutions)	15 multiple-choice questions to solve procedural algorithms and deduction problems	2/2	Pairs outperform solos in 2 design-related tasks.
[Al-Kilidar *et al.*, 05]	Pair-solo (4 metrics of ISO9126)	6 weeks of design tasks of two modules in a web-base network project management tool.	3/8	The effects of pair design on each metric (functionality, usability, portability, and maintainability) are not strong enough to be detected.
[Bellini *et al.*, 05]	Pair-solo (score)	Design improvement for a 15-class system	1/2	The study assesses the impact of educational background on effectiveness of pairs. Only pairs of developers with similar background outperform solos.
[Canfora *et al.*, 06]	Pair-solo (score)	4 evolution design tasks groupped to 2 assignments	3/5	Pairs outperform solos in the second run.
[Canfora *et al.*, 07]	Pair-solo (score)	4 evolution design tasks grouped to 2 assignments	4/5	The experimental design is similar to [Canfora *et al.*, 06], but the subjects are professional developers. Pairs outperform solos in the second run and also outperform on the basis of each assignment.

programming [77], [76], [6]. For the rest of the included studies, the term "case" is equivalent to the hypothesis.

Table 5 presents the outcomes of five experiments related to pair design. All the experiments evaluate the effectiveness of pair and solo design. Out of 22 cases, pair design has been found to be more effective than solo design in 13 cases. The ratio of effective cases is relatively high for most of the pair design experiments, except for the work of Al-Kilidar et al. [1]. This might be caused by the adopted quality metrics that do not strongly reflect the effects of pair design.

The results of 12 studies that evaluate performance of PP in a colocated environment are presented in Table 6. The effective usage of PP has been detected in 14 out of 35 cases. Only two studies [2], [68] further investigate the impact of confounding factors of PP. In other experiments, the effects of confounding factors are claimed to be eliminated using random assignments.

The effectiveness of distributed PP has been analyzed in eight cases presented in Table 7. Out of all cases, three compare effectiveness of distributed PP with colocated pairs, two compare it with solo programming, one with a distributed team, and one with three other working configurations. Based on these results, distributed pairs appear to perform better than solos, but not better than colocated pairs.

Furthermore, PP has been evaluated with respect to other SE techniques, e.g., formal code inspection. Table 8 presents the results of these studies. The effective usage of PP has been detected only in 3 out of 15 cases. While these studies reveal no evidence that PP performs worse, this low ratio reflects that PP might be relatively comparable to other techniques to enhance software quality.

Table 9 summarizes perceptions of PP from the point of view of industrial practitioners. Two studies report about surveys conducted at Microsoft and at a Finnish software company. A third study reports the feedback of a Finnish PP team. In all the cases, practitioners like PP. They claim that PP improves code quality and spreads knowledge of code and design among team members.

Madeyski [44] evaluates the effectiveness of pairs in writing test scripts. Branch coverage and mutation score (i.e., the ability of tests to detect defects purposefully injected in a code) are used as quality metrics. Pair testing does not appear to be more effective than solo testing in terms of both measures.

According to the existing empirical evidence [81], there is an initial adjustment period in the transition from solo to collaborative programming. During this period the developers are said to "jell" [35]. In industry, this adjustment period takes usually hours or days, depending upon the individuals. At the university, the students generally jell after the first assignment, though some reported an even shorter adjustment period [81]. However, a limited number of studies about PP have included more than one task in the experiment to allow for pair jelling and to explore whether the short-term effects observed in the experiments were representative of long-term development [10], [18], [19], [33], [39], [68], [77], [81].

To conclude, existing studies do not support any significant conclusions on the effects of PP. Its effectiveness depends primarily on the following variables: subjects, context, and performed tasks. However, details of such variables are not always sufficiently presented in these studies. This could be caused by the following reasons:

- In spite of the presence of a framework to perform a research on PP [27], there is no standard for classification of situational variables (e.g., task complexity) so that the results from the studies could be placed in the right situation.
- Important confounding factors are not properly treated.

TABLE 6
Summary of Results for Colocated PP

Study	Treatments (quality metrics)	Task description	Effective cases	Outcomes and comments
[Nosek, 98]	Pair-solo (score)	45 minutes to write a script to perform a database consistency check	1/2	Effects of PP on code functionality and readability are tested. Pairs outperform solos only in terms of functionality.
[Williams *et al.*, 00a]	Pair-solo (passed test cases)	6-week 4 programming assignments	1/1	Pairs outperform solos.
[Hulkko and Abrahamsson, 05]	Pair-solo (defect density, several code metrics)	Intranet or mobile applications of 3700–7700 LOC, 5.2–10 man-months	2/3	Teams composed of pairs do not produce a code with fewer defects. Compared to solos, pairs write more comments in the code and do not strictly follow coding standards.
[Vanhanen and Lassenius, 05]	Pair-solo (defect density, proportion of bad methods)	A 400-hours project to implement a distributed, multi-player casino system. The system sizes vary from 2.6k – 5.5k LOC.	1/2	The code implemented by pairs contains more post-release defects but has lower proportion of "bad methods." These effects might be caused by the difference of the implemented use cases.
[Xu and Chen, 05]	Pair-solo (numbers of missed change propagations)	Implementation of 6 changes on an open-source course management application with 31 classes	1/1	Pairs outperform solos in implementing 6 changes in open source software projects.
[Madeyski, 06]	Pair, solo, pair & TDD, solo & TDD (dependency metrics)	Development of an accounting system containing 27 user stories	0/5	Different development approaches do not impact design quality.
[Müller, 06]	Pair-solo (number of failures)	Design and development of scheduling and control modules for an elevator	0/1	In a design phase all teams are composed of pairs. Then, in a development phase one team continues working in pairs and developers of another team work individually. It has been found that the PP team does not outperform individuals.
[Arisholm *et al.*, 07]	Pair-solo (score)	Change implementation on a simple system (12 classes of delegated design) and a complex system (7 classes of centralized control style)	3/12	Two factors are considered: task complexity and professional expertise. The results show that: 1) junior pairs outperform juniors working alone; 2) pairs outperform solos when working on complex tasks; 3) junior pairs outperform juniors working alone only when working on complex tasks.
[Sison, 08]	Pair-solo (defect density)	Design programming exercises and 2-month development of modules for a UML tutor application	1/1	Pairs outperform solos.
[Sison, 09]	Pair-solo (defect density)	Development work of PSP supporting tool (10-100 KLOC), and very small programs (200 LOC)	1/2	Developers implement small and large programs. Pairs outperform solos only when implementing large programs.
[Phaphoom *et al.*, 11]	Pair-solo (defect density)	Defect corrections and enhancements on a large and complex system	3/5	PP reduces the introduction of new defects when working on defect correction tasks.

- In some studies, the size of the sample is relatively small; thus the effects of PP cannot be detected by statistical tests.
- Varieties of adopted quality metrics result in different interpretations.

In this study, we observe the effectiveness of professional pair programmers in performing maintenance tasks in a large industrial software project. The analysis is based on a quality metric of the corresponding parts of the source code and on the extent to which PP was used. We focus on defect correction and new requirement implementation tasks.

TABLE 7
Summary of Results for Distributed PP on Code

Study	Treatments (quality metrics)	Task description	Effective cases	Outcomes and comments
[Baheti *et al.*, 02]	Pair - solo (score)	A 5-week team project using OOP (to build GUI, to implement a dynamic reviewer-mapping algorithm, to simulate LC-2 architecture)	0/1	Effectiveness of teams working in four different configurations appears to be comparable.
[Stotts *et al.*, 03]	Distributed pair - distributed team (passed test cases)	Implementation of a card game	1/1	Code produced by distributed pairs passes more test cases than code produced by solos.
[Favela *et al.*, 04]	Distributed pair - co-located pair (number of defects)	Bug localization and correction, and implementations of small changes in 3 simple programs for counting LOC, sorting, and calculating area of a figure	0/1	Distributed pairs appear to be as good as co-located pairs in localizing defects.
[Domino *et al.*, 07]	Distributed pair - co-located pair (score)	45-minute tasks creating a pseudo-code to implement a discount invoice module of a invoice processing program, and to generate a sales report for a car dealer.	0/2	Co-located pairs outperform distributed pairs when working on both simple and complex tasks.
[Duque and Bravo, 08]	Distributed pair - solo (score, number of syntax error)	Implementation of 3 small programs to read input from keyboard and to perform some calculations	2/3	Pairs obtain a higher score than solos in two out of three tasks. The code produced by pairs contains less syntax error than the code of solos.

3 Research Design

This work is a case study which is conducted to investigate the effectiveness of PP in an industrial setting. The dataset was collected for 14 months from a large Italian manufacturing company. Our main data sources include: PRO Metrics (PROM) [65], work item tracking system, source control system, and source code. We use the goal-question-metrics (GQM) approach [4] to organize the empirical data analysis. The analysis is divided into two parts: 1) *exploratory* data analysis [74] to observe the relationship between PP and DD in code; and 2) *confirmatory* data analysis to confirm the relationship between PP and DD from a statistical standpoint.

To conduct this case study, we sample from variables that represent the typical situation without having a power to manipulate them [85]. Thus, we take the role of observer. As a consequence, the concepts of a control group, an experimental group, and a control period are not applied to our case.

3.1 Goal-Question-Metrics

We define the goal using the standard GQM template as follows:

- Analyzing *source code*.
- For the purpose of *evaluating it*.
- With respect to *relationship between the amount of PP usage and the quality of code*.
- From the view point of *developers*.
- In the context of *industrial software development projects*.

First, we describe the metrics for the amount of PP and the defects.

- *Percentage of PP* (%PP). Effort spent during PP in accessing or modifying methods changed during the observation period. Instead of the real effort in pairs, we use this ratio as it reflects the portions of solo effort. The observation period is defined according to each research question below in this section. The percent PP is defined by

$$\%\mathrm{PP} = \frac{E_P}{E_T},$$

  where E_P is the PP effort spent in the method during an observation period and E_T is the total effort spent in the method in the same period of time. Thus, the solo effort spent in the method in the same period can be evaluated as follows: $E_S = E_T - E_P$.
- *Defect density*. We use the ratio of defects per lines of code to measure the quality of the code. The benefit of using the DD instead of the absolute number of defects is that it is normalized and

TABLE 8
Summary of Results for PP Compared to Other SE Techniques

Study	Treatments (quality metrics)	Task description	Effective cases	Outcomes and comments
[Nawrocki and Wojciechowski, 01]	PSP, XP without PP, XP with PP (number of defects)	Development of 4 programs (to estimate mean and standard deviation, to calculate linear regression parameters, to count logical lines, to count total LOC and methods). Sizes vary from 150 – 400 LOC	0/1	The effectiveness of all development approaches does not seem to be different.
[Phongpaibul and Boehm, 06]	PP-code inspection (number of major and minor defects, passed test cases, and score)	12-week development of a research resource access control system (students), and web application with SMS modules (professionals)	1/4	Pairs outperform solos only when using the number of major defects as a measure.
[Phongpaibul and Boehm, 07]	PP-code inspection (passed test cases)	13-week development of a tool to collect size of source code, which will be integrated to a USC CodeCount toolset.	0/2	The effectiveness of PP and code inspection appears to be comparable.
[Müller, 04] [Müller, 05] [Müller, 07]	PP-code inspection (number of defects)	Development of programs to find positions of a third degree polynomial, and to find a solution of shuffle-puzzle.	2/8	The study investigates types of defects produced by pairs in two tasks. Pairs produce fewer "specification" defects in the first task, and fewer "expression" defects in the second task.
[Keeling, 10]	PP-code inspection (defect count)	Implementation of features for a web-based tool used during requirement elicitation.	0/1	During a coding phase a PP team detects more defects than a team of individuals that performs code inspection. Though, during acceptance testing more defects are detected in the code produced by pairs.

comparable among methods of different size. Moreover, there is empirical evidence that the total number of defects has a pattern of relationship with lines of code [38]. DD hence reduces such dependence, which might generate bias for the analysis. DD is defined by

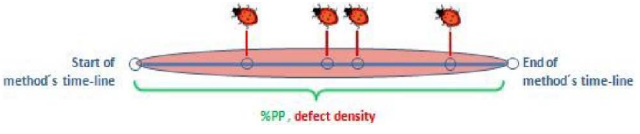
$$DD = \frac{D_T}{LOC},$$

where D_T is the total number of defects found in the method and LOC is the method's Lines of Code. Defects in D_T are part of the 8.4 percent of defects having complete tracking information during the corrections (Section 3.5); the cardinality of D_T changes according to each research question below in this section.

The quality metric adopted in our study is based on the mapping of defects with their original locations in the source code. Since there is no direct linkage, we define a mapping algorithm for this purpose. The quality metric hence relies primarily on the results of the applied algorithm.

We have measured PP usage and the defects in the methods in several situations, aiming at determining the situations in which PP could be more effective than solo programming. We achieved this goal by considering *events which occur during the method's lifetime*. The following cases illustrate this concept.

Case 1: Considering PP usage on defective methods for the whole observation period.



The purpose of the first case is to observe the distribution of DD in a method with respect to the amount of PP. As mentioned, the unit of our analysis is a method (in classes). The data points included in this case are the defective methods that could be mapped to the defects found during the whole observation period (Section 3.5). For each method, %PP is the ratio between the PP effort and the total effort that the developer(s) spent working on that method. DD is calculated by using the total number of defects found and the method's LOC at the end of the observation period.

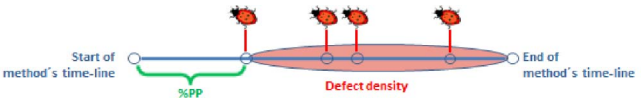
Case 2: Considering the time at which a defect was found in the method.

TABLE 9
Summary of Perceptions of PP in Industry

Study	Setting (quality metrics)	Task description	Effective cases	Detailed outcomes
[Vanhanen and Korpi, 07]	An analysis of a software development project in a large Finnish telecommunication company (defects)	12-week development of an internal reporting system	n/a	After 1.5 years of active production usage only 5 defects were reported. PP is considered to improve design understandability and code quality.
[Vanhanen and Lassenius, 07]	Survey at a medium-sized Finnish software company (voting score)	n/a	n/a	PP is considered to enhance understandability and maintainability of code, to increase user satisfaction, and to reduce defects.
[Begel and Nagappan, 08]	Survey in Microsoft (voting score)	n/a	n/a	2/3 of respondents like PP. 2/3 believe that PP increases software quality.

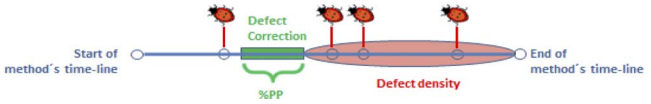
TABLE 10
Summary of Results for Pair Testing

Study	Setting (quality metrics)	Task description	Effective cases	Detailed outcomes
[Madeski, 07]	Branch coverage and mutation score	Writing unit tests using JUnit for testing an accounting system.	0/1	The effectiveness of pair and solo testing does not seem to be different.



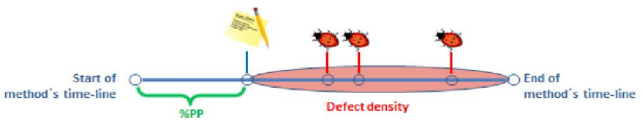
In the second case, we are interested in the amount of PP that has been spent in the method before the identification of the defect. The objective of this observation is to analyze *"whether or not the more effort spent during PP results in the less defect rate in a method."* This analysis could be used to study a long-term effect of PP on defect prevention. This case considers the same methods as the first case. % PP is now given by the ratio between the PP effort and the total effort spent before the defect has been found. DD is calculated by using the total number of defects found afterward. A method containing more than one defect therefore appears in the data points several times with different values of %PP and DD.

Case 3: Considering the duration in which the defect was fixed.



In the third case, we are interested in the amount of PP used during defect correction and the defects found afterward. In general, it is likely that developers would introduce new defects while changing existing code. This analysis would provide evidence of whether it is often the case and whether PP helps to improve the situation. The data points are the same methods used in the first two cases. %PP is now evaluated as the ratio between the PP effort and the total effort spent during the defect correction. DD is calculated by using the total number of defects found after the correction. Similarly to the second case, methods containing more than one defect appear in the data points several times.

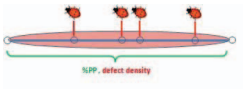
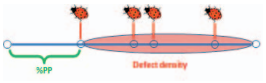
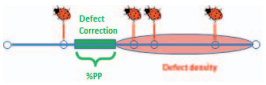
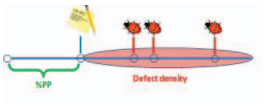
Case 4: Considering the time at which a change is made in the method (the method is changed as a part of user story implementation).

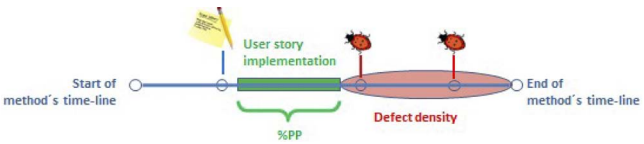


The fourth case focuses on the amount of PP that has been spent in a method before a developer starts to modify it (as a part of the implementation of a user story). This analysis aims at understanding whether or not *regular* usage of PP enhances developers' knowledge over the method (e.g., understand the underlined design decisions and dependencies) and results in better performances when making changes (i.e., introducing none or fewer defects). The data points included in this case are the methods that have been modified during the implementation of user stories. For each data point, %PP is given by the ratio between the PP effort and the total effort spent on that method before a developer starts to modify it. DD is calculated by using the total number of defects found after the implementation has started. A method associated to more than one user story appears in the data points several times with different values of %PP and DD.

Case 5: Considering the duration in which the user story was being implemented.

TABLE 11
Situations under Analysis

Cases	Research question	Parts of code	Focus period of method's lifetime	Assumptions and limitations
Case 1	Is there a relationship between the usage of PP and the defect rate in the code during the whole observation period?	Defective methods		n/a
Case 2	Is there a relationship between the usage of PP and the defect rate in the code?	Defective methods		Other factors than PP might affect the defect rate.
Case 3	Does PP help to reduce the introduction of new defects when it is used for defect correction?	Defective methods		We assume that defects found after the correction were introduced during the change.
Case 4	Does the enhanced knowledge over the code caused by the regular usage of PP help to reduce defects?	Changed methods		Other factors than PP might affect the defect rate.
Case 5	Does PP help to reduce the introduction of new defects when it is used during the implementation of user stories?	Changed methods		We assume that defects found after the implementation were introduced during the change.



The last case focuses on the implementation of a user story. We measure the amount of PP used in this duration and the amount of defects found afterward. This observation could be used to understand the effectiveness of PP in implementing a user story. In this case, the same set of methods as the fourth case is considered. %PP is now given by the ratio between the PP effort and the total effort spent during the implementation. DD is calculated by using the total number of defects found after the implementation has been completed. Similarly to the fourth case, methods associated to more than one user story appear in the data points several times.

Table 11 summarizes the five cases under study. It describes the research question corresponding to each case, the focused parts of code, the focused event or duration in the method's lifetime, and the assumptions/limitations.

For these five cases, we formulate the null hypothesis as follows:

H_0: There is no significant difference between a method accessed entirely during solo programming and a method accessed during PP with respect to DD.

This hypothesis applies to all five cases stated above. For each case, we consider a different set of methods and duration of PP as described in Table 11.

3.2 Exploratory and Confirmatory Data Analysis

We perform an exploratory data analysis to observe the structure of the effects of PP in the five considered contexts. We apply the following models:

- Zero inflated Possion Regression.
- Linear regression model.
- Nadaraya-Watson kernel estimation [53], [80].
- Local-polynomial regression (which is a generalization of kernel estimation).
- Cubic smoothing splines [42], [66].

We perform two different nonparametric tests to obtain a valid conclusion from a statistical perspective:

- a Wilcoxon-Mann-Whitney nonparametric test [92];
- a two-samples permutation test [91].

3.3 Context

The case study is based on the 14-months dataset collected from a team of professional developers working in an IT department of a large Italian manufacturing company (that prefers to remain anonymous). The team is composed of 17 developers: 15 veterans and 2 newcomers. The developers are all Italians aged between 30 and 40 years. They all hold university degrees in computer-related areas and have from 10 to 15 years of programming experience. The team works on several projects, mainly in C#. They are an Agile team, using a customized version of XP. In particular, they use weekly iterations, PP, user stories, and the test-first approach. They use PP spontaneously, i.e., they work in pairs when they find it useful and appropriate. They do not plan when to do PP, with whom, or when to switch roles.

TABLE 12
Six Types of Raw Data Used to Prepare the Measures for Hypotheses Testing and Analysis

Data	Description	Source
Effort and working duration	Effort and duration (seconds) devoted to a specific method/class in a source code. This dataset is collected automatically by the tool that runs as a background process on the developer's machine.	PROM
PP configuration and timeframe	Paired programmers, effort, and time spent doing PP on a specific work, and on specific pieces of source code.	PROM – SPM
Workitem - timeframe	Effort and duration of a specific workitem.	PROM – SPM
Workitem tracking information	Details of activities related to a workitem, e.g., status, important dates, and responsible person. In this study we consider three types of workitems: defects, user stories, and tasks.	Workitem tracking system
Change log	Change details of committed files in a version control system.	Source control system
Method status tracking	A status (i.e., added, modified, removed) of each method in a committed class/file in a version control system.	Source code and source control system

The team had been using XP for more than two years previously to our study. The members of the team were not aware of the actual purpose of our study.

All the developers are located in an open workspace to encourage information flow and collaboration inside the team. Each team member is equipped with a personal machine, two monitors, a single keyboard, and a mouse.

3.4 Data and Data Collection

We combined several types of raw data to prepare the metrics. We describe data and sources in Table 12. The dataset for this study was mainly collected from the following four data sources: PROM, work item tracking system, source control system, and source code.

The data that represent developers' activities have been collected by means of PROM [25], [65]. PROM is an automated tool for noninvasive data collection and analysis that runs on background on the developer's machine. It collects a customizable set of product and process measures [16], [26]. The design of PROM is based on a plug-in architecture that allows the tool to be integrated into a development environment and office applications. Therefore, developers are not distracted from their daily work during the data collection process.

The dataset collected by PROM is composed of a series of time frames that represent developers' interaction over the source code and a series of PP activities. Each interaction consists of a timestamp, a name of a currently focused method or a class, the duration, and the identifiers (id) of the paired developers if they do PP during the time frame.

The story point manager (SPM) is a plug-in of PROM that is responsible for linking the class methods with the work items that are accessed by the developers when working on a particular work item—a unit of work in a project. By means of this plug-in, the developers can specify the work item they are currently working on. PROM automatically records the effort devoted to each work item. In this study, we classify the work items into three categories: defects, user stories, and tasks. Possible states of a work item are as follows: created, assigned, resolved, verified, and closed. Furthermore, we developed an application to identify the parts of code that the developers changed when working on a specific work item. We use this information to analyze the effectiveness of PP when working on different categories of work items.

3.4.1 Data Reliability and Privacy

We consider the data collected by PROM very reliable for the following reasons:

- Developers were familiar with the PROM interface and PP plug-in. They were using the tool for a couple of months before we started collecting data.
- The dataset for this study was collected noninvasively. This way, the developers were not distracted from their daily work and they did not change their usual behavior because of the ongoing study.
- Every day during the whole study period the developers received time reports containing the percentage of time devoted to each application in the previous day. Before the beginning of the study we asked the developers to check the data very carefully and to report any inconsistency. The developers confirmed the correctness of the collected data.

The dataset collected by PROM is highly confidential. The developers were informed in detail about PROM and what kind of data it collects. Moreover, the collected data were first stored on the developer's machine. The developers could access their own dataset and decide whether to send it to the central database or to delete. Furthermore, it is important to mention that the participation in the study was on a voluntary basis [15].

Regarding data completeness, during our study none of the developers decided to entirely stop the data collection. Nevertheless, as explored by Coman et al. [14], there could

TABLE 13
Summary of Work Item-Methods Mapping

4 steps to map workitems and changes in source code	Remaining number of workitems		
	Defects (the total number is 464)	User stories (the total number is 1635)	Tasks (the total number is 111)
1. Sufficient tracking information was available.	430	1568	90
2. + 3. It was possible to identify the part of code that was *'accessed'* during the work.	88	274	0
4. It was possible to identify the part of code that was *'modified'* during the work.	39	144	0
Total workitems for further analysis	8.4%	8.8%	0%

be an internal limitation of the tool that collected data. While the tool is able to discern long idle periods at the computer and eliminate them from the total time recorded (thus from the data considered in our study), it is not able to discern short interruptions (seconds or a few minutes). This type of interaction can be easily generated since the developers work in a shared environment where informal communication is frequent [14].

3.5 Mapping Defects to Source Code

To measure DD in each class method, we map defects to their locations in a source code. For this purpose, we identify the methods that were changed during the defect correction activity. We perform the following steps for such mapping:

1. *Identification of defects with sufficient information.* Preliminary conditions are thus defined as
   a. Information during the defect life cycle was sufficiently recorded. The essential information included: the defect id, the title, the state, the reason (resolution), responsible person of each state change, and timestamps of each state change.
   b. The final status of the defect correction was "complete" and it introduced changes in the source code.
2. *Identification of a sequence of time frames when developers were working on a particular defect.* The input to this step was the list of events generated by PROM—SPM. The tool generated and sent an event to PROM server when a work session on a specific work item was started, paused, resumed, and stopped. As such, for each defect, the sequence of defect correction time frames (T_1-T_n) could be determined.
3. *Identification of a list of methods or classes being accessed during defect correction.* The input to this step was the defect correction time frame (T_1-T_n) and the developer-source code interaction data collected by PROM. By comparing work time on PROM data set, we identified the list of methods (M_1-M_n) invoked during each time frame. This list contained methods through which a developer was browsing during defect localizations and corrections. PROM did not record whether an access to a file was to read such file or to modify it. However, such information was essential for this study. Therefore, the next step identified the methods that were accessed and modified during the defect correction.
4. *Identification of a subset of methods being modified during the defect corrections.* The input to this step was the change log from the source control system. This data provided a list of files and classes that were changed for each commit. Using the API of the source control system, we generated a program to track the state of a method when its belonging file or class changed.

The goal of this algorithm is to identify the list of methods modified during defect correction and the time frames in which the developer worked on those methods. The second and the third step are necessary to identify a collection of time frames and methods associated to each work item. The fourth step is needed to distinguish those methods modified for defect corrections from those modified for other purposes. Without the information acquired from the third step, it would not be possible to determine the purpose of the changes detected for each commit.

Apart from defect corrections, we applied the same mechanism to identify a list of methods that were changed during the user story implementation and general tasks. Table 13 summarizes the amount of remaining work items after we applied each of the four steps. As a result, there were 8.4 percent of defects with complete tracking information during defect corrections and 8.8 percent of user stories with complete information during the implementation. The general tasks were related to the document, rather than to the code. This information was used for exploratory analysis.

3.6 Descriptive Statistics

To summarize the number of methods associated to the implementation of each work item, Fig. 3 presents the total number of methods being accessed and modified for the defect corrections and the implementation of the user stories. In total, 377 methods were modified for 39 defect corrections and 1,904 methods were modified for 144 user story implementations.

As mentioned above, we were able to map 8.4 percent of defects found during the observation period to the source code. Fig. 4 shows how these defects are distributed over the 377 methods. The maximum number of defects per method is 4; only three methods (0.8 percent) correspond to

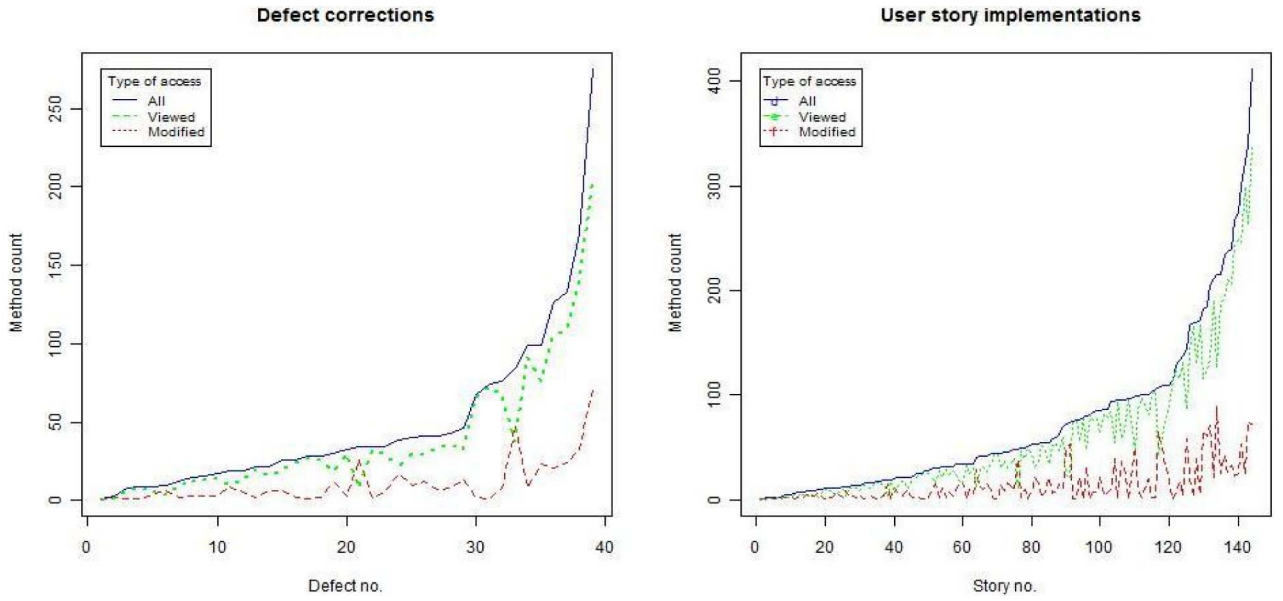


Fig. 3. Number of methods associated to each work item, defect correction, and user story implementation.

this case. Four methods (1.06 percent) contain three defects; 18 methods (4.77 percent) contain two defects. The majority of the methods (352, 93.37 percent) contain only one defect.

In the company, each user story is associated with a certain priority given by the customer. This value represents the level of importance to the customer; the team works on the user stories in priority order, meaning that the high-priority stories are implemented first. To represent the priority of both the user stories and the defects, an ordinal scale (from 1 to 3) is used, 1 indicating the highest priority. Table 14 shows the descriptive statistics of the priority of defects and user stories. A total of 213 out of 1,568 user stories hold priority 1; 1,351 have priority 2; only four user stories have priority 3. Out of 430 defects with sufficient information for analysis (Section 3.5), 51 defects hold priority 1; 378 defects hold priority 2; and only one defect has priority 3. The analysis of the relationship between the priority (of defects and user stories) and the time needed to complete the work is presented in Section 5 (case 3 for defect corrections and case 5 for user story implementations).

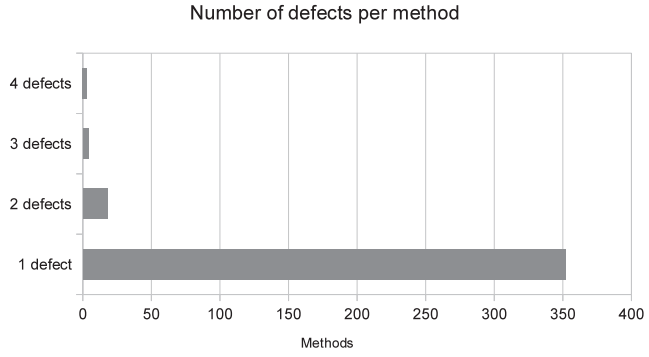


Fig. 4. Distribution of defects over the 377 methods.

4 Analysis Approach

4.1 Exploratory Analysis

One of the most widely used general statistical procedures to study the relationship between a response variable and a set of explanatory ones is the linear (in the parameters) regression:

$$y_i = \beta_0 + \beta_1 x_{1i} + \cdots + \beta_p x_{pi} + \epsilon_i, \quad i = 1, \ldots, n, \qquad \text{(M1)}$$

with the errors usually taken to be independent and identically (often, normally) distributed with zero mean and constant variance. If this model is a good representation of reality, least squares estimations of β_i can be calculated and inference and prediction follow. Fitting a linear model to a nonlinear relationship can give results that are worse than useless, implying a degree of certainty that is not realistic [31]. The abundance of zeros which characterizes the available data suggests an overall exploratory analysis of software defects and programming approaches (pair and solo programming) using a Zero-inflated Poisson regression (ZIPR), a model count data that has an excess of zero counts. The excess zeros are supposed to be generated by a separate process from the count values and they modeled independently from the other count data. After the identification of a positive (as an algebraic sign) relationship between defects and solo programming and negative one between defects and PP, multiple nonparametric regressions have been run to analyze this specific relationship for each of the five research questions. A more general alternative to model M1 is the nonparametric regression model:

$$y_i = M(x_i) + \epsilon_i, \;\; i = 1, \ldots, n. \qquad \text{(M2)}$$

The regression curve $m(x)$ is the conditional expectation $m(x) = E(Y|X = x)$, i.e., it does not need to hold exactly for each single observation but only on average. The model M2 removes the parametric restrictions on $m(x)$ and allows (perhaps unexpected) alternative structures to come through.

TABLE 14
Priority of Work Items

Workitems	Priority 1	Priority 2	Priority 3	Total
Defects	51 (11.86 %)	378 (87.90 %)	1 (0.23 %)	430 (100%)
User stories	213 (13.58 %)	1351(86.16 %)	4 (0.26%)	1568 (100%)

In this study, to explore the structure of the effects of PP in the five contexts in which the usual assumptions required to apply a traditional linear model do not hold, three of the most common nonparametric regression models have been applied:

- Nadaraya-Watson kernel estimation [53], [80].
- Local-polynomial regression (which is a generalization of kernel estimation).
- Cubic smoothing splines [42], [66].

The Nadaraya-Watson method, introduced and studied in Watson [80] and Nadaraya [53], provides a smoothing estimate of the regression function $m(x)$ by local weighted averaging of the y_i values:

$$m_{NW}(x) = \frac{\sum_{i=1}^{n} K\left(\frac{x-x_i}{h}\right) y_i}{\sum_{i=1}^{n} K\left(\frac{x-x_i}{h}\right)} = \sum_{i=1}^{n} w_i y_i. \quad \text{(M3)}$$

A linear function of the response values y_i where

- $\mathrm{K} : \mathrm{R} \rightarrow \mathrm{R}$ is a bounded, integrable function (e.g., Uniform, Epanechnikov, Quadratic, Gaussian). Additionally, K is often positive and null everywhere, except on a compact subset of R;
- h is the bandwidth parameter controlling the smoothness of the estimate.

Local polynomial regression is similar to kernel estimation, but the fitted values are produced by locally weighted regression to fit a dth degree polynomial to data rather than by locally weighted averaging ($d = 0$).

Cubic smoothing splines are the solution to the penalized regression problem [89]; find m to minimize

$$S(h) = \sum_{i=1}^{n} [y_i - f(x_i)]^2 + h \int_{x_{(1)}}^{x_{(n)}} [f''(x)]^2 dx. \quad \text{(M4)}$$

Here, h is a roughness penalty, analogous to the span in nearest-neighbor kernel or local polynomial regression, and $f''(x)$ is the second derivative of the regression function (taken as a measure of roughness). Without the roughness penalty, nonparametrically minimizing the residual sum of squares would simply interpolate the data. The mathematical basis for smoothing splines is more satisfying than for kernel or local polynomial regression since an explicit criterion of fit is optimized, but spline and local polynomial regressions of equivalent smoothness tend to be similar in practice.

4.2 Confirmatory Analysis

To obtain a valid conclusion from a statistical perspective, we perform two different nonparametric tests:

- a Wilcoxon-Mann-Whitney non parametric test [92], and
- a two-samples permutation test [91].

The Wilcoxon-Mann-Whitney test checks the null hypothesis that the distributions from which two samples are drawn do not differ by a location shift against the alternative that they differ by some location shift. This test (which brings results comparable to the Mann-Whitney U test) is used when 1) the data are in a rank order format since it is the only format in which scores are available, or 2) the data have been transformed into a rank order format from an interval/ratio format since the researcher has reason to believe that the normality assumption, as well as the homogeneity of variances assumption of the Student's t test for two independent samples, is violated.

The Wilcoxon-Mann-Whitney test is based on the following assumptions [92]:

1. each sample is randomly selected from the population it represents;
2. the two samples are independent;
3. the original variable observed (which is subsequently ranked) is a continuous random variable; and
4. the underlying distributions from which the samples are derived are identical in shape.

Taking into account how the experiment has been designed, assumptions 1 and 2 hold. Assumption 3 is very common in nonparametric tests, although it is generally violated [92]. In this work, it can be considered met, as DD in the code are counts over the total lines of code, i.e., it is a discrete random variable; since the lines of code are a very large number, DD can be assumed to be a continuous variable with a good degree of approximation. Assumption 4 has been tested through the use of a two samples Kolmogorov-Smirnov test on standardized samples to check if the two subsets of each group can be considered to be equal in shape.

The permutation test [91] is a type of statistical significance test in which the distribution of the test statistic under the null hypothesis is obtained by calculating all possible values of the test statistic under rearrangements of the labels on the observed data points. The process is identical to that of Bootstrap method except that in permutation test the selection of data is done without replacing. The basic premise is to use only the assumption that it is possible that all of the treatment groups (PP and solos) are equivalent and that every member of them is the same before sampling began. These conditions hold as all the projects are equally likely to suffer defects before the observation period. For two data sample sets $x = x_1, x_2, \ldots, x_n$ and $y = y_1, y_2, \ldots, y_n$, the process starts by adding the two data samples together to get a new sample set w from which two new samples x_w and y_w of size n and m, respectively, are sampled without

TABLE 15
Zero-Inflated Poisson Regression Output

	Coef.	z	P>z	[95% Conf.	Interval]
Solo programming	.0001461	8.28	0.000	.0001115	.0001807
Pair programming	-.0002742	-2.75	0.006	-.0004699	-.0000785
Constant	-5.438349	-31.05	0.000	-5.781685	-5.095012
Inflate: Constant	.1121442	0.64	0.522	-.2310613	.4553497

replacement. The difference of the two sample means $d_w = \bar{x}_w - \bar{y}_w$ is computed and the process repeated a number of times (say k times). The final result is the distribution of the k differences between the k couples of permuted samples. Using this distribution, it is possible to compute the p-value on the true difference $d = \bar{x} - \bar{y}$ between the two sample means. If d is a typical value of the aforesaid simulated distribution, the two samples can be considered to be equal in mean. If it is an atypical value, the difference between the two samples cannot be linked to a random assignment of units to the two samples. The two samples are different because they come from two different distributions as it is very unlikely that samples coming from the same distribution are significantly different in mean only on a random basis.

5 Results of Exploratory Analysis

To have an overlook on the relationship among the software defects and pair or solo programming, a preliminary ZIPR has been done (significance of the model: $\mathrm{Prob} > \chi^2 = 0.001$). The model, expressing the absolute software defects out of the LOC (exposure variable) in dependence of the pair and solo programming absolute efforts, points out (Table 15) that whereas solo programming has an incremental effect on software defects (the corresponding coefficient is positive and significant as $\mathrm{P} > \mathrm{z}$ is smaller than 0.001), PP has a decremental effect on them (the corresponding coefficient is negative and significantly different from zero).

After this general result, and to explore the relationship between the defects density and the PP level, the five research questions have been analyzed using the aforesaid nonparametric regression models. As nonparametric regressions are not based on parametric assumptions, they have to be viewed as exploratory data analysis techniques in which the role of data is of primary relevance so that the usual inferential procedures do not make sense.

Case 1: Considering PP usage on defective methods for the whole observation period. The purpose of Case 1 is to explore the relationship between PP practiced on defective methods and the defect rates. Here, we consider 377 methods that were modified for a purpose of the defect correction. The scatter plot (Fig. 5) shows the DD of those methods against the %PP. %PP is zero for those methods that were created and modified totally during solo programming sessions. In this case, 50 methods out of 377 were modified in total during solo sessions.

Over the whole observation period (Fig. 5 (left-side)), the DD observed does not seem to have a systematic decrease over higher levels of PP practice. The nonparametric interpolation of data through the use of nonparametric estimates (Fig. 5 (right-side) shows, on the contrary, a counterintuitive behavior with a tendency of increase of defects over PP practice. The reason for this particular behavior is due to the overabundance of zero-cooperation projects, which determines a low conditional mean of defects for PP practice (%PP) equal to zero. Looking at data, it is in fact possible to identify the highest rates of defects (above 0.3) for solo programming.

Case 2: Considering the time at which a defect was found in the method. The purpose of Case 2 is to explore the relationship between PP practiced prior to the detection of a defect and the defect rate. Here, we assume that this fraction of PP practice would reflect the enhanced knowledge over the

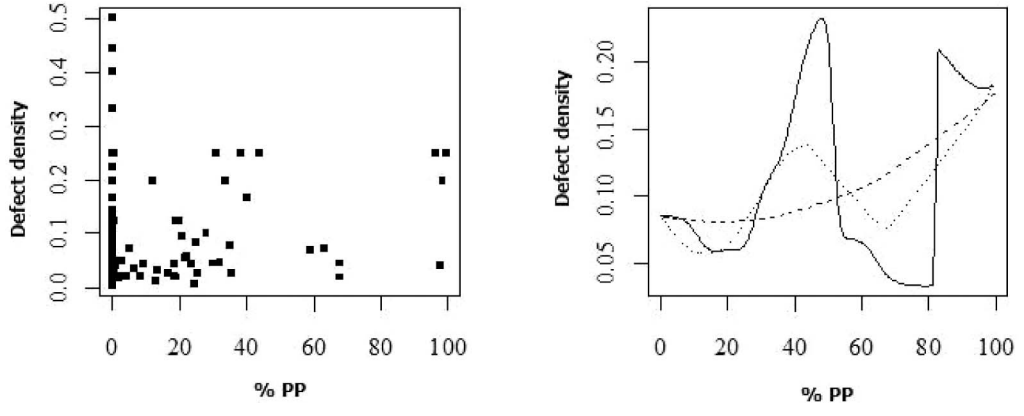


Fig. 5. Data analysis of Case 1—the whole observation period. Data points (left side) and nonparametric regression models (right side: kernel (bandwidth = 12) continuous line, local polynomial (span = 10) dashed line, spline (degrees of freedom = 5) dotted line).

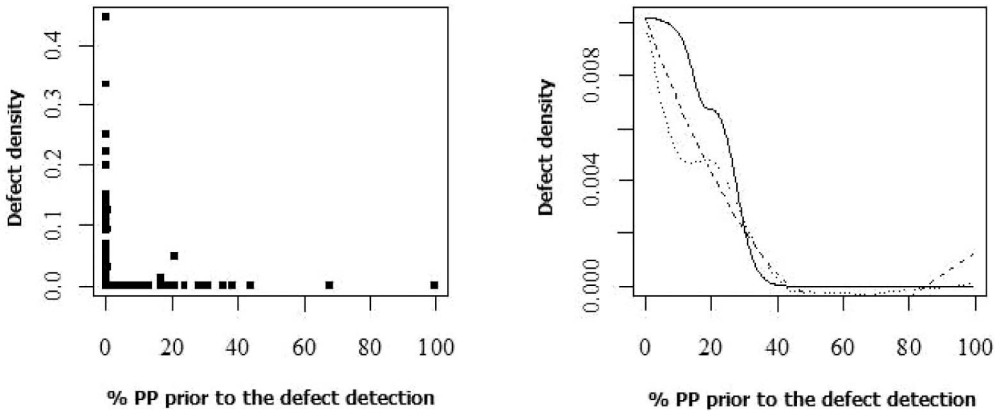


Fig. 6. Data analysis of Case 2—PP practice prior to the detection of the defect. Data points (left side) and nonparametric regression models (right side: kernel (bandwidth = 15) continuous line, local polynomial (span = 10) dashed line, spline (degrees of freedom = 5) dotted line).

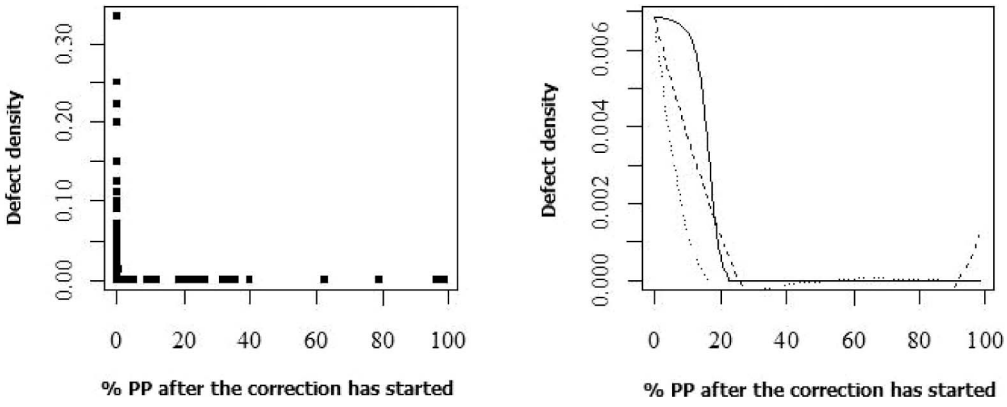


Fig. 7. Data analysis of Case 3—PP practiced after the defect correction has started. Data points (left side) and nonparametric regression models (right side: kernel (bandwidth = 15) continuous line, local polynomial (span = 10) dashed line, spline (degrees of freedom = 5) dotted line).

source code. Three hundred seventy-seven methods modified for the defect corrections results in 412 data points as some of the considered methods contain more than one defect. Out of 412, 32 data points have %PP equal to zero.

Considering this fraction of PP practice, it can be stated that there is a tendency in the defect reduction over the higher %PP (Fig. 6 (left-side)). All three models (Fig. 6 (right-side)) show this evident decrease for %PP lower than approximately 30 %PP. It is interesting to note that even if the two nonparametric models, local polynomial and spline, are very "flexible" they tend to specify unrealistically negative response values for %PP higher than 50 percent. The Kernel model assumes only positive numbers.

Case 3: Considering the duration in which the defect was fixed. The purpose of Case 3 is to explore the relationship between PP practiced during the defect correction and the introduction of new defects. We use the number of defects that were discovered after the correction was started to determine the effectiveness of PP in correcting defects. Out of 412 considered data points, 26 data points have %PP equal to zero.

In this configuration, after the defect correction has started, DD of methods drop quickly even for low levels of PP. Although local polynomial and spline models assume negative values for cooperation levels (%PP) bigger than 20 percent, their steep descent before those values emphasizes the impression given from data in Fig. 7 (left-side).

Table 16 shows the results of the tests performed to find any effect of the priority of the defects on the time to fix. The test is weakly significant ($\text{p-level} = 0.0438$) stating that the difference between the two times to fix is present but not very strong. This is also confirmed by the two-samples Kolmogorov-Smirnov test to check if the two samples (low and high priority) are drawn from the same continuous (realistically a Gamma) distribution which is weakly significant with a $\text{p-value} = 0.08$. Therefore, it can be stated that the defect time to fix is not connected or is weakly connected to the bug priority.

Case 4: Considering the time at which a change was made in the methods during an implementation of a user story. In the same manner as Case 2, Case 4 is designed to explore the relationship between PP practiced prior to the enhancement on methods and defect rate. Here, we analyze PP practiced on 1,904 methods, which were modified during the implementation of user stories. Out of 1,904, 242 methods have %PP equal to zero.

The tendency in defect reduction found in Q2 can be confirmed herein. Except for a peak of the Kernel function at approximately 50 %PP, the defect rate is decreasing constantly as %PP grows. Moreover, it can be seen in Fig. 8 (right-side) how all three models remain "high" as defects are present for values of % PP lower than 20 percent. The peak cited above is due to the high flexibility of the kernel method in comparison with the local polynomial or spline models.

TABLE 16
Effect of the Priority of the Defects on the Time to Fix

	Group 1*		Group 2**			Total	
Valid N	51		379			430	
Mean (minutes)	680		466			491	
Gamma shape parameters	0.27155		0.27155				
Gamma rate parameters	0.00066		0.00058				
Mann-Whitney U Test							
	Rank Sum Group 1	**Rank Sum Group 2**	**U**	**Z**	**P-level**	**Valid N Group 1**	**Valid N Group 2**
Time to fix (hours)	12.670	79.995	7.985	2.015707	0.043831	51	379
Two-sample Kolmogorov-Smirnov test							
D = 0.189; p.value = 0.08048							
* priority 1; ** priority 2 and priority 3							

Case 5: Considering the duration in which the user story was being implemented. In the same manner as Case 3, Case 5 (Fig. 9) explores the relationship between PP practiced after the implementation of user stories has started and the introduction of new defects. We used the number of new defects discovered after this time to determine the effectiveness of PP in performing enhancements on the system. In this case, out of 1,904 methods, 378 have percent PP equal to zero.

The relationship between the DD and the cooperation level has a behavior very similar to Case 3. In this case, the local regression model has not be estimated as it gave inconsistent results due to the structure of data.

Table 17 shows the results of the tests performed to explore the effect of the priority of the user stories on the time to implement. The test is strongly significant ($\text{p-level} < 0.001$) stating that the difference between the two times is evident. This is also confirmed by the two-samples Kolmogorov-Smirnov test to check if the two samples (low and high priority) are drawn from the same continuous (realistically Gamma) distribution, which is strongly significant with a $\text{p-value} < 0.001$. Therefore, it can be stated that the user story time to implement is strongly connected with the priority.

6 Results of Confirmatory Analysis

If PP has direct and relevant effects on the number of defects in software development, a difference between PP and solo programming should be observed. To this end, all five datasets have been split into two groups according to the presence of pair or solo programming. To test whether the two groups represent populations with different mean values of code defects, two different nonparametric tests have been performed: a Wilcoxon-Mann-Whitney nonparametric test and a two-samples permutation test. The choice of using two deeply different tests has been made to support the results achieved in a robust and reliable perspective.

The results of these tests (including the Kolmogorov-Smirnov test for the equality of the PP and solo continuous distributions to satisfy the Wilcoxon-Mann-Whitney test assumption) along with some key figures of the datasets are given in Table 18. There is a slight effect of PP on defects as all group mean differences are negative. At any rate, the significance of these differences is very low as all the p-values, both the Wilcoxon-Mann-Whitney test and the Permutation test, are fairly above the significance level of 0.1. The p-values obtained from the two tests are different

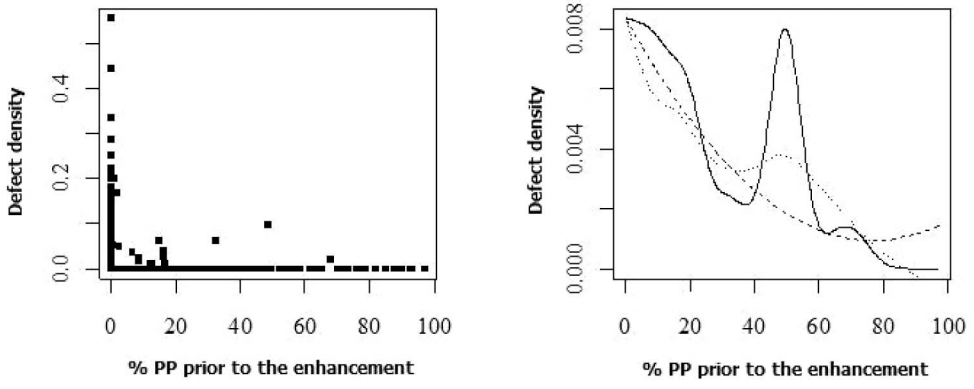


Fig. 8. Data analysis of Case 4—PP practiced before the enhancement. Data points (left side) and nonparametric regression models (right side: kernel (bandwidth = 15) continuous line, local polynomial (span = 10) dashed line, spline (degrees of freedom = 5) dotted line).

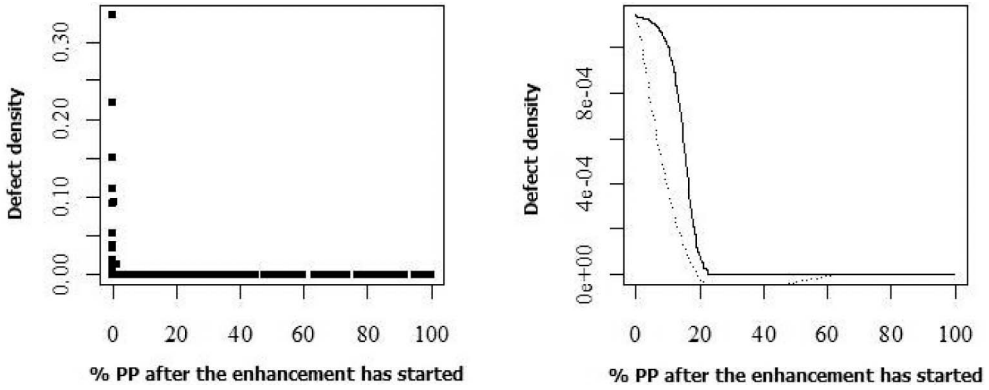


Fig. 9. Data analysis of Case 5—After the enhancement has started. Data points (left side) and nonparametric regression models (right side: kernel (bandwidth = 15) continuous line, spline (degrees of freedom = 5) dotted line).

TABLE 17
Effect of the Priority of the User Stories on the Time to Fix

	Group 1*		Group 2**			Total	
Valid N	213		1,355			1,568	
Mean (minutes)	1,294		726			803	
Gamma shape parameters	0.68573		0.47458				
Gamma rate parameters	0.00053		0.00065				
Mann-Whitney U Test							
	Rank Sum Group 1	**Rank Sum Group 2**	**U**	**Z**	**P-level**	**Valid N Group 1**	**Valid N Group 2**
Time to fix (hours)	199208.00	1030888.00	112198.00	-5.23	0.00	213	1,355
Two-sample Kolmogorov-Smirnov test							
D = 0.1839; p.value = 7.852e-06							
* priority 1; ** priority 2 and priority 3							

(but this should not be surprising, as they are two very different approaches to test similar hypotheses). However, they move in the same direction: PP does not seem to give a drastic and significant effect in mean defects reduction.

Table 19 summarizes the results obtained with the exploratory analysis and with the confirmatory analysis.

7 Validity

A fundamental question concerning the results of a case study is how valid the results actually are. Adequate validity requires that the results should be valid for the population of interest [85]. This section explains what we consider to be the most important validity issues of this case study (Table 20).

7.1 Construct Validity

The construct validity is concerned with the relation between the theory and the observation [85]. We must ensure that the treatment reflects the construct of the cause and the outcome reflects the construct of the effect.

PP. PP has some facets regarding how pairs are constructed and how pairs work. In this study, the team of developers has experience in PP. They work in a colocated environment and PP is a general practice in the team. PROM collected (automatically) the information regarding pair construction, duration, and the pieces of source code the pairs worked on. We received a confirmation from developers that they were constantly using the tool. Therefore, we consider the measure of %PP of methods very reliable. However, we could not capture the amount of PP practice across the whole life cycle of methods since sometimes a portion of it lies outside our observation time frame. The change in its values hence affected the results and the analysis, though the duration of our data collection lasted approximately 14 months. We consider this duration sufficient to reflect the actual amount of PP practice.

Quality measure. We use DD to measure the quality of methods instead of using the absolute number of defects. The possible bias of using the absolute number of defects could be caused by incomparability of methods of different size. We use DD to eliminate the bias. Instead of using DD to measure code quality, different code metrics (e.g., CK metric and cyclomatic complexity metrics) could be used. However, this information was not available in our case.

Defects. Issues related to defects include: 1) The missing evidence of defect corrections could have changed the value of the outcome (DD); 2) the changes classified as defect corrections might have been caused by other purposes; and

TABLE 18
PP and Solo Group Statistics and Tests for the Equality of Two Group Means

Case	Means of the defect density			Number of observations		P-values		
	PP	Solo	Difference between the means	PP	Solo	Kolmogorov-Smirnov-test	Wilcoxon-Mann-Whitney test	Permutation test
Case 1	.0840	.0862	-.0022	327	50	0.798	0.690	0.855
Case 2	.0097	.0098	-.0001	380	32	0.999	0.192	0.984
Case 3	.0005	.0065	-.0060	386	26	0.999	0.809	0.312
Case 4	.0050	.0080	-.0030	1662	242	0.994	0.282	0.209
Case 5	.0007	.0010	-.0003	1526	378	0.999	0.137	0.711

TABLE 19
Results of the Exploratory Analysis and of the Confirmatory Analysis

Case	Research question	Exploratory analysis	Confirmatory analysis
Case 1	Is there a relationship between the usage of PP and the defect rate in the code during the whole observation period?	The defect density *does not have a systematic decrease* over higher levels of PP practice. The non-parametric interpolation shows a counterintuitive behavior with a tendency of increase of defects over PP practice.	There is a slight effect of PP on defects as all group mean differences are negative. The significance of these differences could not be confirmed because all the p-values, both of the Wilcoxon-Mann-Whitney test and Permutation test, are fairly above the significance level of 0.1. The results of both the tests move in the same direction: PP appears to provide a perceivable but small effect on the reduction of defects in these settings.
Case 2	Is there a relationship between the usage of PP and the defect rate in the code considering how much PP was devoted to a method until a defect was detected?	There is *a tendency in the defect reduction* over the higher % PP. The non-parametric interpolation shows an evident decrease for cooperation levels lower than approximately 30% PP.	
Case 3	Does PP help to reduce the introduction of new defects when it is used for defect correction?	After the defect correction has started, defect density of methods *drop quickly even for low levels of PP*. The descent of defect density is evident when PP is bigger than 20%.	
Case 4	Does the enhanced knowledge over the code caused by the regular usage of PP help to reduce defects?	The tendency in defect reduction found in Case 2 can be herein confirmed. The defect rate decreases constantly as PP grows except for a peak of one module at approximately around 50% PP.	
Case 5	Does PP help to reduce the introduction of new defects when it is used during the implementation of user stories?	The relationship between the defect density and the cooperation level has a behavior similar to Case 3.	

3) the remaining defects in the code might have changed the value of DD and, as a result, changed the analysis.

One of the challenges of this research was to map defects with their original location in the code. Some evidence of the defect localization activities (e.g., series of time frames in which the developer works on a particular task) was not automatically collected by PROM. Instead, it was generated by a feature that needed to be activated manually when developers started working on specific defects. If it was not activated, then the evidence would be missing. This could change the value of some metrics, i.e., the number of methods modified to correct the defect, absolute defects, and DD. Moreover, the result of the program that maps defects to source code (Section 3.5) could be affected by the completeness of this working time frame information.

The changes that have been counted as defects in a method's DD are supposed to be due to defect corrections. This is because the tool that generates and sends an event to the PROM server was started by the developer when he/she was starting a work session on a specific work item.

Regarding the remaining defects, since some time has passed between the writing of the code and the data extraction, we can suppose that most of the defects have been found. Given that the observation lasted 14 months, the number of remaining defects could be such a small number that they would not affect the analysis.

TABLE 20
Summary of the Threads to Validity

Construct validity	Internal validity	External validity
- Some portion of PP usage could lie outside our observation time frame - Code quality is measured by defect density instead of using an absolute number of defects or code metrics - The missing evidences of defect corrections could affect the value of defect density - The changes classified as defect corrections might have been caused by other purposes - Undiscovered defects might affect the analysis	- We do not have information on other confounding factors that might have an impact on code quality rather than PP usage	- Even though there is tendency that PP might improve code quality, the results from non-parametric tests are not significant

7.2 Internal Validity

Internal validity concerns with the causal relationship between the treatment and the outcome. If the relationship between them is observed, we must be certain that it is not a result of other factors over which we have no control or which we have not measured [85].

The problem of confounding factors is a crucial issue for an observational case study where we have no control over the subjects and their practices. In PP context, two crucial confounding factors to the effectiveness of pairs are the expertise of developers and task complexity [2]. In our case study, all the developers are professionals: 15 veterans and two newcomers. We found no significant effects of the two types of developers using multiple regression analysis. Task complexity depends upon the complexity of the system and the difficulty of the task. We were not able to test the effect of this factor due to the limitations of the dataset.

7.3 External Validity

The external validity is concerned with generalization. If the relationship between the construct of the cause and the effect exists, threads to external validity are analyzed to determine if the result could be generalized outside the scope of the study [85].

In our study, the statistical analysis reveals that there is a consistent and slight negative effect of PP on defects, even though the significance of these differences is very low. When considering the conditions that could limit our ability to generalize the results (i.e., three types of interaction with the treatment: selection, setting, and time [85]), the study holds strength conditions from the following perspectives: First of all, the subjects are a good representation of the population we want to generalize, since the data comes from a team of 17 professional software developers. The better case is to use stratified samples. To date, most studies on PP have been conducted with students, and it is still unclear how well results from student-based experiments could be generalized to professional developers. Second, the study is conducted in an industrial environment in which the subjects naturally perform their daily tasks on a large and complex system. The experiments in which the subjects perform simple and independent development tasks and follow informal development processes might not represent industrial environments very well. Third, our study covers a large time frame—14 months. This allows us to investigate the long-term effect of PP and also removes the threat from interaction of time and treatment.

8 Conclusions and Future Work

The case study was conducted to evaluate the effect of PP on quality and efficiency of defect corrections. The dataset was collected from an Agile software development project at a large Italian manufacturing company. The data collection covered a time frame of 14 months. Compared with existing case studies on PP, the use of long periodical data in real software development context makes the study more realistic.

In our exploratory analysis, the effectiveness of PP was investigated in the context of defect corrections and implementations of user stories. The analysis shows that the introduction of new defects tends to decrease when PP is practiced. The results are consistent for both contexts. Even though the significance of this behavior could not be confirmed by the nonparametric tests, PP appears to provide a perceivable but small effect on defect reduction in these settings.

Further studies on PP should extend the scope of the present study in three ways. First, our results suggest that PP helps to reduce the DD. However, the number of defects to the case study was relatively small. Further case studies should include more defects. Second, the analysis should take into account confounding factors, programmer expertise, complexity of tasks, and geographical dispersion [34], [70]; moreover, it would be important to try to understand the root causes of such effects, like developers attention [72], using also innovative analysis techniques [17], [58]. Third, an analysis of the percentage of programming compared to other activities has shown that pair programmers devote more time to programming activities than solo programmers [24]. The integration of the dataset used in this study with those in [24] also using suitable analysis techniques [71] would serve to investigate how the percentage of programming affects the DD.

REFERENCES

[1] H. Al-Kilidar, P. Parkin, A. Aurum, and R. Jeffery, "Evaluation of Effects of Pair Work on Quality of Designs," *Proc. Australian Conf. Software Eng.,* pp. 78-87, 2005.

[2] E. Arisholm, H. Gallis, T. Dyba, and D.I.K. Sjoberg, "Evaluating Pair Programming with Respect to System Complexity and Programmer Expertise," *IEEE Trans. Software Eng.,* vol. 33, no. 2, pp. 65-86, Feb. 2007.

[3] P. Baheti, E.F. Gehringer, and P.D. Stotts, "Exploring the Efficacy of Distributed Pair Programming," *Proc. Second XP Universe and First Agile Universe Conf. Extreme Programming and Agile Methods—XP/Agile Universe,* pp. 208-220, 2002.

[4] V. Basili, "Applying the Goal Question Metric Paradigm in the Experience Factory," *Proc. 10th Ann. Conf. Software Metrics and Quality Assurance in Industry,* 1993.

[5] K. Beck, *Extreme Programming Explained: Embrace Change.* Addison-Wesley Professional, Oct. 1999.

[6] A. Begel and N. Nagappan, "Pair Programming: What's in It for Me?" *Proc. ACM-IEEE Second Int'l Symp. Empirical Software Eng. and Measurement,* pp. 120-128, 2008.

[7] E. Bellini, G. Canfora, A. Cimitile, F. Garcia, M. Piattini, and C. Visaggio, "The Impact of Educational Background on Design Knowledge Sharing during Pair Programming: An Empirical Study," *Professional Knowledge Management,* K.-D. Althoff, A. Dengel, R. Bergmann, M. Nick, and T. Roth-Berghofer, eds., vol. 3782, pp. 455-465, Springer, 2005.

[8] T. Bipp, A. Lepper, and D. Schmedding, "Pair Programming in Software Development Teams—An Empirical Study of Its Benefits," *Information Software Technology,* vol. 50, no. 3, pp. 231-240, 2008.

[9] G. Braught, J. MacCormick, and T. Wahls, "The Benefits of Pairing by Ability," *Proc. 41st ACM Technical Symp. Computer Science Education,* 2010.

[10] G. Canfora, A. Cimitile, C.A. Visaggio, F. Garcia, and M. Piattini, "Performances of Pair Designing on Software Evolution: A Controlled Experiment," *Proc. European Conf. Software Maintenance and Reeng.,* pp. 197-205, 2006.

[11] G. Canfora, A. Cimitile, F. Garcia, M. Piattini, and C.A. Visaggio, "Evaluating Performances of Pair Designing in Industry," *J. Systems Software,* vol. 80, no. 8, pp. 1317-1327, 2007.

[12] J. Chong and T. Hurlbutt, "The Social Dynamics of Pair Programming," *Proc. 29th Int'l Conf. Software Eng.,* pp 354-363, 2007.

[13] A. Cockburn and L. Williams, *The Costs and Benefits of Pair Programming,* pp. 223-243. Addison-Wesley Longman Publishing Co., Inc., 2001.

[14] I.D. Coman, A. Sillitti, and G. Succi, "Investigating the Usefulness of Pair-Programming in a Mature Agile Team," *Proc. Ninth Int'l Conf. eXtreme Programming and Agile Processes in Software Eng.,* pp. 10-14, 2008.

[15] I.D. Coman, A. Sillitti, and G. Succi, "A Case-Study on Using an Automated In-Process Software Engineering Measurement and Analysis System in an Industrial Environment," *Proc. 31st Int'l Conf. Software Eng.,* pp. 89-99, 2009.

[16] I. Coman, A. Sillitti, and G. Succi, "Ensuring Continuous Data Accuracy in AISEMA Systems," *Proc. 23rd Int'l Conf. Software Eng. and Knowledge Eng.,* July 2011.

[17] E. Di Bella, A. Sillitti, and G. Succi, "A Multivariate Classification of Open Source Developers," *Information Sciences,* vol. 221, no. 1, pp. 72-83, Feb. 2013.

[18] M.A. Domino, R.W. Collins, and A.R. Hevner, "Controlled Experimentation on Adaptations of Pair Programming," *Information Technology and Management,* vol. 8, no. 4, pp. 297-312, 2007.

[19] R. Duque and C. Bravo, "Analyzing Work Productivity and Program Quality in Collaborative Programming," *Proc. Third Int'l Conf. Software Eng. Advances,* pp. 270-276, 2008.

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

[21] R.L. Edwards, J.K. Stewart, and M. Ferati, "Assessing the Effectiveness of Distributed Pair Programming for an Online Informatics Curriculum," *ACM Inroads,* vol. 1, pp. 48-54, 2010.

[22] J. Favela, H. Natsu, C. Prez, O. Robles, A.L. Morn, R. Romero, A.M. Martnez-Enrquez, and D. Decouchant, "Empirical Evaluation of Collaborative Support for Distributed Pair Programming," *Groupware: Design, Implementation and Use,* G.-J. de Vreede, L.A. Guerrero, and G. Marn Ravents, eds., pp. 215-222, Springer, 2004.

[23] I. Fronza, A. Sillitti, and G. Succi, "An Interpretation of the Results of the Analysis of Pair Programming during Novices Integration in a Team," *Proc. Third Int'l Symp. Empirical Software Eng. and Measurement,* pp. 225-235, 2009.

[24] I. Fronza, A. Sillitti, G. Succi, and J. Vlasenko, "Analysing the Usage of Tools in Pair Programming Sessions," *Proc. Int'l Conf. Agile Processes and eXtreme Programming in Software Eng.,* pp. 1-11, 2011.

[25] I. Fronza, A. Sillitti, G. Succi, and J. Vlasenko, "Understanding How Novices Are Integrated in a Team Analysing Their Tool Usage," *Proc. Int'l Conf. Software and Systems Processes,* May 2011.

[26] I. Fronza, A. Sillitti, G. Succi, M. Terho, and J. Vlasenko, "Failure Prediction Based on Log Files Using Random Indexing and Support Vector Machines," *J. Systems and Software,* vol. 86, no. 1, pp. 2-11, Jan. 2013.

[27] H. Gallis, E. Arisholm, and T. Dyba, "An Initial Framework for Research on Pair Programming," *Proc. Int'l Symp. Empirical Software Eng.,* p. 132, 2003.

[28] B. Hanks, "Student Performance in CS1 with Distributed Pair Programming," *Proc. 10th Ann. SIGCSE Conf. Innovation and Technology in Computer Science Education,* pp. 316-320, 2005.

[29] J. Hannay and M. Jorgensen, "The Role of Deliberate Artificial Design Elements in Software Engineering Experiments," *IEEE Trans. Software Eng.,* vol. 34, no. 2, pp. 242-259, Mar./Apr. 2008.

[30] J.E. Hannay, E. Arisholm, H. Engvik, and D.I.K. Sjoberg, "Effects of Personality on Pair Programming," *IEEE Trans. Software Eng.,* vol. 36, no. 1, pp. 61-80, Jan./Feb. 2010.

[31] W. Härdle, *Smoothing Techniques.* Springer-Verlag, 1991.

[32] S. Heiberg, U. Puus, P. Salumaa, and A. Seeba, "Pair-Programming Effect on Developers Productivity," *Proc. Int'l Conf. Agile Processes and eXtreme Programming in Software Eng.,* 2003.

[33] H. Hulkko and P. Abrahamsson, "A Multiple Case Study on the Impact of Pair Programming on Product Quality," *Proc. 27th Int'l Conf. Software Eng.,* pp. 495-504, 2005.

[34] A. Janes, B. Russo, P. Zuliani, and G. Succi, "An Empirical Analysis of the Discontinuous Use of Pair Programming," *Proc. Fourth Int'l Conf. Extreme Programming and Agile Processes in Software Eng.,* pp. 205-214, 2003.

[35] J.R. Katzenbach and D.K. Smith, *The Wisdom of Teams: Creating the High-Performance Organization.* Harper Business, 1994.

[36] M. Keeling, "Put It to the Test: Using Lightweight Experiments to Improve Team Processes," *Proc. 11th Int'l Conf. Agile Processes in Software Eng. and Extreme Programming,* vol. 48, pp. 287-296, 2010.

[37] B. Kitchenham, "Procedures for Performing Systematic Reviews," technical report, Keele Univ. and NICTA, 2004.

[38] A.G. Koru, D. Zhang, K. El Emam, and H. Liu, "An Investigation into the Functional Form of the Size-Defect Relationship for Software Modules," *IEEE Trans. Software Eng.,* vol. 35, no. 2, pp. 293-304, Mar./Apr. 2009.

[39] K.M. Lui, K.C.C. Chan, and J. Nosek, "The Effect of Pairs in Program Design Tasks," *IEEE Trans. Software Eng.,* vol. 34, no. 2, pp. 197-211, Mar./Apr. 2008.

[40] K.M. Lui, K.A. Barnes, and K.C.C. Chan, "Pair Programming: Issues and Challenges," *Proc. Int'l Conf. Agile Software Development: Current Research and Future Directions.* pp. 143-163, 2010.

[41] K.M. Lui and K.C.C. Chan, "When Does a Pair Outperform Two Individuals?" *Proc. Fourth Int'l Conf. Extreme programming and agile Processes in Software Eng.,* pp. 225-233, 2003.

[42] Q. Li and J.S. Racine, *Nonparametric Econometrics: Theory and Practice.* Princeton Univ. Press, Nov. 2006.

[43] L. Madeyski, "The Impact of Pair Programming and Testdriven Development on Package Dependencies in Objectoriented Design—An Experiment," *Proc. Seventh Int'l Product-Focused Software Process Improvement,* vol. 4034, pp. 278-289, 2006.

[44] L. Madeyski, *On the Effects of Pair Programming on Thoroughness and Fault-Finding Effectiveness of Unit Tests,* J. Münch, P. Abrahamsson, eds., pp. 207-221. Springer, 2007.

[45] E. Mendes, L. Al-Fakhri, and A. Luxton-Reilly, "A Replicated Experiment of Pair-Programming in a Second-Year Software Development and Design Computer Science Course," *Proc. 11th Ann. SIGCSE Conf. Innovation and Technology in Computer Science Education,* pp. 108-112, 2006.

[46] R. Moser, M. Scotto, A. Sillitti, P. Abrahamsson, and G. Succi, "Does XP Deliver Quality and Maintainable Code?" *Proc. Eighth Int'l Conf. eXtreme Programming and Agile Processes in Software Eng.,* pp. 18-22, 2007.

[47] C. McDowell, L. Werner, H.E. Bullock, and J. Fernald, "The Impact of Pair Programming on Student Performance, Perception and Persistence," *Proc. 25th Int'l Conf. Software Eng.,* pp. 602-607, 2003.

[48] L. Murphy, S. Fitzgerald, B. Hanks, R. McCauley, "Pair Debugging: A Transactive Discourse Analysis," *Proc. Sixth Int'l Workshop Computing Education Research,* 2010.

[49] M.M. Müller, "Are Reviews an Alternative to Pair Programming?" *Empirical Software Eng.,* vol. 9, no. 4, pp. 335-351, 2004.

[50] M.M. Müller, "Two Controlled Experiments Concerning the Comparison of Pair Programming to Peer Review," *J. Systems and Software,* vol. 78, no. 2, pp. 166-179, 2005.

[51] M.M. Müller, "A Preliminary Study on the Impact of a Pair Design Phase on Pair Programming and Solo Programming," *Information and Software Technology,* vol. 48, 335-344, May 2006.

[52] M.M. Müller, "Do Programmer Pairs Make Different Mistakes than Solo Programmers?" *J. Systems and Software,* vol. 80, no. 9, pp. 1460-1471, 2007.

[53] E.A. Nadaraya, "On Estimating Regression," *Theory of Probability and Its Applications,* vol. 9, no. 1, 141-142, 1964.

[54] N. Nagappan, L. Williams, M. Ferzli, E. Wiebe, K. Yang, C. Miller, and S. Balik, "Improving the CS1 Experience with Pair Programming," *Proc. 34th SIGCSE Technical Symp. Computer Science Education,* pp. 359-362, 2003.

[55] J. Nawrocki and A. Wojciechowski, "Experimental Evaluation of Pair Programming," *Proc. European Software Control and Metrics Conf.,* 2001.

[56] A. Nickel and T. Barnes, "Games for CS Education: Computer-Supported Collaborative Learning and Multiplayer Games," *Proc. Fifth Int'l Conf. Foundations of Digital Games,* pp. 274-276, 2010.

[57] J.T. Nosek, "The Case for Collaborative Programming," *Comm. ACM,* vol. 41, no. 3, 105-108, 1998.

[58] W. Pedrycz, G. Succi, P. Musilek, and X. Bai, "Using Self-Organizing Maps to Analyze Object-Oriented Software Measures," *J. Systems and Software,* vol. 59, no. 1, 65-82, 2001.

[59] N. Phaphoom, A. Sillitti, and G. Succi, "Pair Programming and Software Defects—An Industrial Case Study," *Proc. Int'l Conf. Agile Processes in Software Eng. and Extreme Programming,* vol. 77, pp. 208-222, 2011.

[60] M. Phongpaibul and B. Boehm, "An Empirical Comparison between Pair Development and Software Inspection in Thailand," *Proc. ACM/IEEE Int'l Symp. Empirical Software Eng.,* pp. 85-94, 2006.

[61] M. Phongpaibul and B. Boehm, "A Replicate Empirical Comparison between Pair Development and Software Development with Inspection," *Proc. First Int'l Symp. Empirical Software Eng. and Measurement,* pp. 265-274, 2007.

[62] A.D. Radermacher and G.S. Walia, "Investigating the Effective Implementation of Pair Programming: An Empirical Investigation," *Proc. 42nd ACM Technical Symp. Computer Science Education,* pp. 655-660, 2011.

[63] N. Salleh, E. Mendes, J. Grundy, and G.St.J. Burch, "The Effects of Neuroticism on Pair Programming: An Empirical Study in the Higher Education Context," *Proc. ACM-IEEE Int'l Symp. Empirical Software Eng. and Measurement.* 2010.

[64] N. Salleh, E. Mendes, J. Grundy, and G.S.J Burch, "An Empirical Study of the Effects of Conscientiousness in Pair Programming Using the Five-Factor Personality Model," *Proc. ACM/IEEE 32nd Int'l Conf. Software Eng.,* vol. 1, pp. 577-586, 2010.

[65] A. Sillitti, A. Janes, G. Succi, and T. Vernazza, "Collecting, Integrating and Analyzing Software Metrics and Personal Software Process Data," *Proc. 29th Conf. EUROMICRO,* p. 336, 2003.

[66] J.S. Simonoff, *Smoothing Methods in Statistics.* Springer, May 1998.

[67] R. Sison, "Investigating Pair Programming in a Software Engineering Course in an Asian Setting," *Proc. 15th Asia-Pacific Software Eng. Conf.,* pp. 325-331, 2008.

[68] R. Sison, "Investigating the Effect of Pair Programming and Software Size on Software Quality and Programmer Productivity," *Proc. 16th Asia-Pacific Software Eng. Conf.,* pp. 187-193, 2009.

[69] D. Stotts, L. Williams, L. Williams, P. Baheti, P. Baheti, D. Jen, D. Jen, A. Jackson, and A. Jackson, "Virtual Teaming: Experiments and Experiences with Distributed Pair Programming," technical report, 2003.

[70] A. Sillitti, A. Janes, G. Succi, and T. Vernazza, "Measures for Mobile Users: An Architecture," *J. System Architecture,* vol. 50, no. 7, pp. 393-405, 2004.

[71] A. Sillitti, G. Succi, and J. Vlasenko, "Toward a Better Understanding of Tool Usage (NIER Track)," *Proc. 33rd Int'l Conf. Software Eng.,* 2011.

[72] A. Sillitti, G. Succi, and J. Vlasenko, "Understanding the Impact of Pair Programming on Developers Attention: A Case Study on a Large Industrial Experimentation," *Proc. 34th Int'l Conf. Software Eng.,* 2012.

[73] G. Succi, W. Pedrycz, M. Marchesi, and L.A. Williams, "Preliminary Analysis of the Effects of Pair Programming on Job Satisfaction," *Proc. Third Int'l Conf. Extreme Programming,* pp. 212-215, 2002.

[74] J.W. Tukey, *Exploratory Data Analysis.* Addison-Wesley, 1977.

[75] T. VanDeGrift, "Coupling Pair Programming and Writing: Learning about Students' Perceptions and Processes," *Proc. 35th SIGCSE Technical Symp. Computer Science Education,* pp. 2-6, 2004.

[76] J. Vanhanen and C. Lassenius, "Perceived Effects of Pair Programming in an Industrial Context," *Proc. 33rd EUROMICRO Conf. Software Eng. and Advanced Applications,* pp. 211-218, 2007.

[77] J. Vanhanen and H. Korpi, "Experiences of Using Pair Programming in an Agile Project," *Proc. 40th Ann. Hawaii Int'l Conf. System Sciences,* p. 274b, 2007.

[78] J. Vanhanen and C. Lassenius, "Effects of Pair Programming at the Development Team Level: An Experiment," *Proc. Int'l Symp. Empirical Software Eng.,* pp. 336-345, 2005.

[79] L.G. Votta, "By the Way, Has Anyone Studied Any Real Programmers, Yet?" *Proc. Ninth Int'l Software Process Workshop,* pp. 93-95, 1994.

[80] G. Watson, "Smooth Regression Analysis," *Sankhya,* vol. 26, pp. 359-372, 1969.

[81] L.A. Williams, R.R. Kessler, W. Cunningham, and R. Jeffries, "Strengthening the Case for Pair Programming," *IEEE Software,* vol. 17, no. 4, pp. 19-25, July 2000.

[82] L.A. Williams, A. Shukla, and A.I. Anton, "An Initial Exploration of the Relationship between Pair Programming and Brooks' Law," *Proc. Agile Development Conf.,* pp. 11-20, 2004.

[83] L.A. Williams and R.R. Kessler, "All I Really Need to Know about Pair Programming I Learned in Kindergarten," *Comm. ACM,* vol. 43, no. 5, pp. 108-114, 2000.

[84] L.A. Williams and R.R. Kessler, *Pair Programming Illuminated.* Addison-Wesley Longman Publishing Co., Inc., 2002.

[85] C. Wohlin, P. Runeson, M. Host, M.C. Ohlsson, B. Regnell, and A. Wesslen, *Experimentation in Software Engineering: An Introduction.* Kluwer Academic Publishers, 2000.

[86] S. Xu and X. Chen, "Pair Programming in Software Evolution," *Proc. Canadian Conf. Electrical and Computer Eng.,* pp. 1846 -1849, May 2005.

[87] S. Xu and V. Rajlich, "Empirical Validation of Test-Driven Pair Programming in Game Development," *Proc. Fifth IEEE/ACIS Int'l Conf. Computer and Information Science and First IEEE/ACIS Int'l Workshop Component-Based Software Eng., Software Architecture and Reuse,* pp. 500-505, 2006.

[88] N.Z. Zacharis, "Measuring the Effects of Virtual Pair Programming in an Introductory Programming Java Course," *IEEE Trans. Education,* vol. 54, no. 1, pp. 168-170, Feb. 2011.

[89] A.W. Bowman and A. Azzalini, *Applied Smoothing Techniques for Data Analysis.* Oxford Univ. Press, 1997.

[90] P. Berander, "Using Students as Subjects in Requirements Prioritization," *Proc. 2004 Int'l Symp. Empirical Software Eng.,* pp. 167-176, Aug. 2004.

[91] P.I. Good, *Resampling Methods: A Practical Guide to Data Analysis.* Springer, 2001.

[92] D.J. Sheskin, *Parametric and Nonparametric Statistical Procedures.* CRC, 2000.

[93] W. Harrison, "N = 1: An Alternative for Software Engineering Research," *Proc. 22nd Int'l Conf. Software Eng.,* Beg, Borrow, or Steal: Using Multidisciplinary Approaches in Empirical Software Eng. Research Workshop, 2000.

[94] M. Höst, B. Regnell, and C. Wohlin, "Using Student as Subjects—A Comparative Study of Students and Professionals in Lead-Time Impact Assessment," *Empirical Software Eng.*, vol. 5, no. 3, pp. 201-214, 2000.

Enrico di Bella received the PhD degree in applied statistics from the University of Padua, Italy, in 2005. He is an assistant professor in statistics at the University of Genoa, Italy. His current research interests include data mining and statistics applied to developer's classification and behavior in the software development process.

Ilenia Fronza received the PhD degree in computer science from the University of Bozen-Bolzano, Italy, in 2012. She is currently a research fellow on a fixed-term contract at the Free University of Bolzano-Bozen, Italy. Her current research interests include agile methodologies, data mining, and computational intelligence in software engineering, failure prediction, noninvasive measurement, software process measurement and improvement.

Nattakarn Phaphoom received the double master's degree in software engineering from the Blekinge Institute of Technology, Sweden, and the Free University of Bolzano-Bozen, Italy. She is currently working toward the PhD degree at the Free University of Bolzano-Bozen, Italy. Prior to studying the master's degree, she worked for three years for IBM Solutions Delivery Co., Ltd., Thailand. Her research interests include software process assessment and improvement, agile methods, and IT innovation. Her current research focuses on cloud computing and cloud software development in an innovative context.

Alberto Sillitti received the PhD degree in electrical and computer engineering from the University of Genoa, Italy, in 2005, and the PEng degree. He is an associate professor on the Faulty of Computer Science at the Free University of Bolzano-Bozen, Italy. He has been involved in several EU funded projects related to open source software, services architectures, and agile methods in which he applies noninvasive measurement approaches. He has served as a member of the program committees for several international conferences and as program chair of OSS '07, XP '10, and XP '11. His research areas include open source development, agile methods, software engineering, noninvasive measurement, mobile and web services. He is the author of more than 80 papers published in international conferences and journals. He is a member of the IEEE.

Giancarlo Succi is a professor with tenure at the Free University of Bolzano-Bozen, Italy, where he directs the Centre for Applied Software Engineering. Before joining the Free University of Bolzano-Bozen, he was a professor with tenure at the University of Alberta, Edmonton, Alberta, Canada, an associate professor at the University of Calgary, Alberta, and an assistant professor at the University of Trento, Italy. His research interests include multiple areas of software engineering, including open source development, agile methodologies, experimental software engineering, software engineering over the Internet, and software product lines and software reuse. He is a Fulbright Scholar. He is a member of the IEEE.

Jelena Vlasenko received the bachelor's degree in computer science from the Free University of Bolzano-Bozen, Italy, in 2010. She is currently working toward the master's degree at the Free University of Bolzano-Bozen, Italy, where she is working on her thesis at the Center for Applied Software Engineering. Her current research interests include agile methodologies, noninvasive measurement, software process measurement and improvement.