# Evaluating Pair Programming with Respect to System Complexity and Programmer Expertise

Erik Arisholm, *Member*, *IEEE*, Hans Gallis, Tore Dybå, *Member*, *IEEE Computer Society*, and Dag I.K. Sjøberg, *Member*, *IEEE*

**Abstract**—A total of 295 junior, intermediate, and senior professional Java consultants (99 individuals and 98 pairs) from 29 international consultancy companies in Norway, Sweden, and the UK were hired for one day to participate in a controlled experiment on pair programming. The subjects used professional Java tools to perform several change tasks on two alternative Java systems with different degrees of complexity. The results of this experiment do not support the hypotheses that pair programming in general reduces the *time required* to solve the tasks correctly or increases the proportion of *correct solutions*. On the other hand, there is a significant 84 percent increase in *effort* to perform the tasks correctly. However, on the more complex system, the pair programmers had a 48 percent increase in the proportion of correct solutions but no significant differences in the time taken to solve the tasks correctly. For the simpler system, there was a 20 percent decrease in time taken but no significant differences in correctness. However, the moderating effect of system complexity depends on the programmer expertise of the subjects. The observed benefits of pair programming in terms of correctness on the complex system apply mainly to juniors, whereas the reductions in duration to perform the tasks correctly on the simple system apply mainly to intermediates and seniors. It is possible that the benefits of pair programming will exceed the results obtained in this experiment for larger, more complex tasks and if the pair programmers have a chance to work together over a longer period of time.

**Index Terms**—Empirical software engineering, pair programming, extreme programming, design principles, control styles, object-oriented programming, software maintainability, quasi-experiment.

## 1 Introduction

The concepts underlying pair programming (PP) are not new [13], [15], [17], [40], but PP itself has only recently attracted significant attention and interest within the software industry and academia. Much of the focus on PP is due to the introduction of extreme programming (XP), in which PP is one of 12 key practices [4], [5].

Basically, in PP, two programmers work on the same task using one computer and one keyboard [4], [5], [44], [46]. There are two distinct roles that contribute to a synergy of the individuals in the pair: 1) a driver, who types at the keyboard and focuses on the details of the coding, and 2) a navigator, who actively observes the work of the driver, looking for tactical and strategic defects, thinking of alternatives, writing down "things-to-do," and looking up references. In addition to coding, PP also involves other phases of the software development process, such as design and testing.

Several previous controlled experiments have concluded that PP has many benefits over individual programming, including significant improvements in functional correctness and various other measures of quality of the programs being developed, and reduced duration (a measure of time to market), with only minor additional overhead in terms of total programmer hours (a measure of cost or effort) [25], [30], [32], [44], [46]. One exception is an experiment that showed no positive effects of PP with respect to time taken and no improved functional correctness of the software product compared with individual development [31], which essentially doubled the cost of development. However, the results of that experiment also suggested that the standard deviation of the development times and program sizes of the PP group was lower, suggesting that PP might be more predictable than individual programming.

Most of the existing studies cannot be compared directly, due to differences in sample populations (e.g., students or professionals), study settings (e.g., amount of training in PP), lack of power (e.g., few subjects), and different ways of treating the dependent variables (e.g., how correctness was measured and whether measures of development time also included rework) [19], [30], [31]. However, a common feature of the existing studies is that they have not accounted for the moderating effect of the complexity of the programming tasks, which, in turn, may depend on the complexity of the system being developed or maintained and the expertise of the programmers. In light of existing research in software engineering [2], [22], [35] and social psychology [6], [7], [18], [23], [50], we expected that system complexity and programmer expertise would have a significant impact on *when* and *how* PP is beneficial compared with individual programming [19]. To investigate these issues empirically, we conducted a quasi-experiment that addressed the following research question:

- *E. Arisholm, H. Gallis, and D.I.K. Sjøberg are with Simula Research Laboratory, PO Box 134, NO-1325 Lysaker, Norway. E-mail: {erika, hansga, dagsj}@simula.no.*
- *T. Dybå is with Simula Research Laboratory and SINTEF Information and Communication Technology, NO-7465 Trondheim, Norway. E-mail: tore.dyba@sintef.no.*

*Manuscript received 7 Apr. 2006; revised 7 Sept. 2006; accepted 23 Oct. 2006; published online 28 Dec. 2006.*
*Recommended for acceptance by W.B. Frakes.*
*For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number TSE-0086-0406.*

*What is the effect regarding duration, effort, and correctness of pair programming for various levels of system complexity and programmer expertise when performing change tasks?*

The dependent variables *duration*, *effort*, and *correctness* represent, respectively, one dimension of more general concepts such as time to market, costs, and quality.

Previous experiments on PP have been conducted with students [30], [31], [44] or with a few professionals [25], [32] (both experiments with five pairs and five individuals). To have sufficient power to investigate our research question and to obtain a relatively representative sample of Java programmers, we conducted a two-phase experiment with a total sample of 295 Java consultants (98 pairs and 99 individuals) from 29 consultancy companies in Norway, Sweden, and the UK, including international companies such as Accenture, Cap Gemini, CIBER, Oracle, Steria, and TietoEnator. The first phase of the experiment was conducted on individual developers in 2001 [2]; the second phase was conducted on developer pairs in the second half of 2004 and the first half of 2005. First, all subjects performed a pretest task, the results of which were used to adjust for skill differences between these two groups, e.g., due to the time gap. Subsequently, the subjects performed change tasks on two alternative Java systems based on a centralized or delegated control style, respectively [47].

The remainder of this paper is organized as follows: Section 2 describes the research model of the experiment and the hypotheses. Section 3 gives details of the experimental design and its execution. Section 4 reports the results of testing the study's hypotheses. Section 5 discusses possible threats to validity. Section 6 relates the results to existing research on PP, while Section 7 provides concluding remarks and suggestions for further research.

## 2 Conceptual Model and Hypotheses

Existing research in software engineering has illustrated clearly how the complexity of a programming task may depend on the expertise of the subjects who perform it [2], [22], [35]. For example, the study reported in [35] revealed substantial differences in how novices, intermediates, and experts perceive the difficulties of object-orienteddevelop;-ment. These results were confirmed by a controlled experiment that identified a strong interaction between the expertise of the subjects and the type of task during object-oriented program comprehension [8]. Similar results were found in controlled experiments to assess the effects of design patterns [39] and control styles [2] on maintainability.

Results from social psychology suggest that similar moderating effects may apply to PP. For example, the performance of groups depend on, among other things, the complexity of the tasks [6], [7], [18], [23], [50]. For simple tasks, individuals might perform better than groups. For complex tasks, groups might benefit from the competence of their peers, thus resulting in increased performance compared with individuals.

The conceptual model tested in this study is shown in Fig. 1. In the model, the effects (in our case given by *duration*, *effort*, and *correctness* of the maintained program) of *PP* (versus individual programming) will depend on the moderating variables *system complexity* and *programmer expertise*, both of which will have an impact on the perceived complexity of programming tasks. The conceptual model is motivated by our initial framework for research on pair programming [19]. An overview of the study's hypotheses is presented in Table 1.

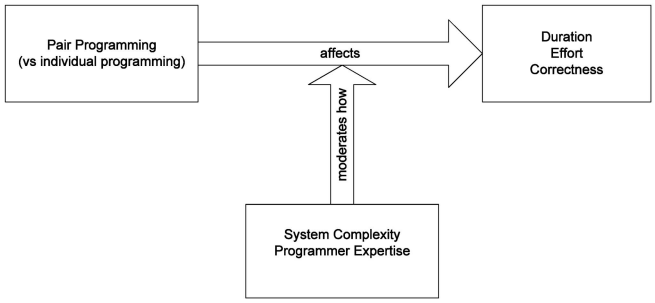



Fig. 1. Conceptual research model of the hypothesized effects of pair programming.

The existing experiments on the effect of PP on duration, effort, and quality were conducted on initial *development* tasks with both students and professionals [25], [30], [31], [32], [44], [46]. In this experiment, we address *change* tasks and professional developers, and propose the null hypotheses $H0_1$, $H0_4$ and $H0_7$ (Table 1) to test the main effect of PP on, respectively, duration, effort, and correctness.

To our knowledge, no existing studies on PP have been designed to assess directly how the complexity of the systems and the programmers' expertise affect the relative performance of pairs versus individuals. To test the moderating effect of system complexity on PP, we propose the null hypotheses $H0_2$, $H0_5$, and $H0_8$ (Table 1). To test the moderating effect of programmer expertise on PP, we propose the null hypotheses $H0_3$, $H0_6$, and $H0_9$ (Table 1).

## 3 Design of the Experiment

The conceptual research model discussed in Section 2 was implemented by means of a two-phase controlled experiment. In the first phase, conducted in 2001 and reported as a separate experiment [2], we evaluated the effect of a centralized versus delegated control style in a Java application for different categories of developer. A total of 158 subjects took part, divided into 59 students (undergraduate and graduate) and 99 professional Java consultants (junior, intermediate, and senior). To compare the performance of pairs with that of individuals, we conducted phase two of the experiment in the second half of 2004 and the first half of 2005 with pairs only (196 Java consultants constituting 98 pairs), using the exact same experimental procedure and material as in the first phase.

The design of the experiment is shown in Fig. 2. Since the subjects were assigned randomly only to the two control style treatments, not to the two pair programming treatments, this is a quasi-experiment [14]. To address threats generated by the nonrandom assignment to the pair

TABLE 1
Informal Descriptions of the Null Hypotheses

| | |
|---|---|
| $H0_1$ | **Effect of Pair Programming on Duration:** The time taken to perform change tasks is equal for individuals and pairs. |
| $H0_2$ | **Moderating Effect of System Complexity on Duration:** The difference in the time taken to perform change tasks for pairs versus individuals does not depend on system complexity. |
| $H0_3$ | **Moderating Effect of Programmer Expertise on Duration:** The difference in the time taken to perform change tasks for pairs versus individuals does not depend on programmer expertise. |
| $H0_4$ | **Effect of Pair Programming on Effort:** The effort expended on performing change tasks is equal for individuals and pairs. |
| $H0_5$ | **Moderating Effect of System Complexity on Effort:** The difference in the effort expended on performing change tasks for individuals and pairs does not depend on system complexity. |
| $H0_6$ | **Moderating Effect of Programmer Expertise on Effort:** The difference in the effort expended on performing change tasks for pairs versus individuals does not depend on programmer expertise. |
| $H0_7$ | **Effect of Pair Programming on Correctness:** The correctness of the maintained programs is equal for individuals and pairs. |
| $H0_8$ | **Moderating Effect of System Complexity on Correctness:** The difference in the correctness of the maintained programs for pairs versus individuals does not depend on system complexity. |
| $H0_9$ | **Moderating Effect of Programmer Expertise on Correctness:** The difference in the correctness of the maintained programs for pairs versus individuals does not depend on programmer expertise. |

programming treatments, the experiment included a common pretest programming task. The results of this test were used to adjust for differences between all treatment groups in an Analysis of Covariance (ANCOVA) model [14], [28]. The combined experiment was a $2 \times 2 \times 3$ factorial design [49] with the following factors:

**Pair Programming:** The pairs in this experiment consisted of two individuals with a similar level of programmer expertise (junior and junior, intermediate and intermediate, and senior and senior). This choice was motivated by previous studies on PP that reported that pairs consisting of individuals with similar competence levels collaborated more successfully than those with different competence levels [9], [19], [42]. Most of the subjects in this experiment had no experience with PP. Furthermore, they performed maintenance change tasks on a program of which they had no prior knowledge.

**System Complexity:** The first phase of the experiment showed that the delegated control-style design of the given application was, on average, more difficult to change than was the alternative, centralized control-style implementation [2]. Thus, the control styles of the application being changed were used to discriminate between two levels of system complexity. However, note that, as discussed in [2], system complexity should not be viewed in absolute terms; it is relative to the expertise of the subjects.

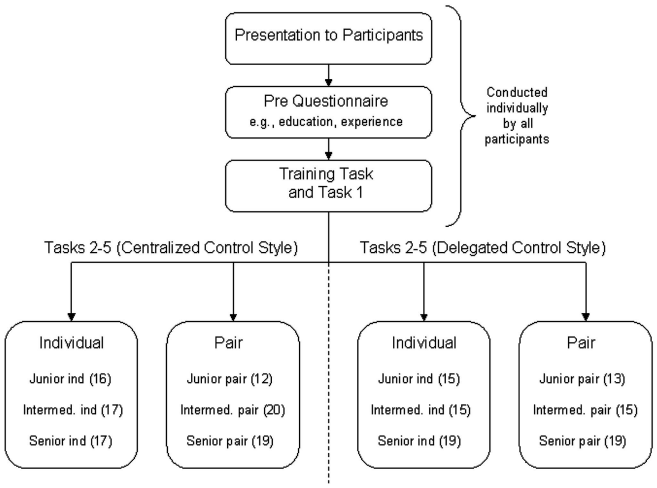



Fig. 2. Experimental design.

**Programmer Expertise:** We used two alternative indicators of programmer expertise. One was programmer category (junior, intermediate, and senior professional Java consultants), as determined by the project managers in the consultancy companies. The other indicator attempted to measure programming skill more directly on the basis of the results of the pretest programming task.

The dependent variables were defined as follows:

**Duration:** Duration was defined as the elapsed time taken to perform a set of change tasks. Since it is not meaningful to compare duration for programs that require rework with programs that do not, we only considered duration for subjects whose work was correct.

**Effort:** Effort was defined as the total number of programmer hours taken to develop a correct program. Thus, effort equals duration for the individuals and twice the duration for the pair programmers. Note that the effort and duration measures are clearly not independent, but we considered both measures to be important as they provide complementary insights of the costs and benefits of pair programming.

**Correctness:** Correctness was defined in terms of whether or not the final, maintained program possessed the required functionality, i.e., a binary score.

Further details on how the dependent variables were measured are provided in Section 3.6.

### 3.1 Power Analysis

Prior to the second phase of the experiment, a power analysis was performed to calculate how many subjects in total ($N$) were needed in the sample. SamplePower 2.0 from

SPSS[1] was used for the power analysis. The alpha level was set to 0.05.

We performed a power analysis for a logistic regression-based test on the basis of the dependent variable correctness and the assumption that there was an equal number of subjects in each category of the main treatment (individuals and pairs). Based on the results from the first phase and previous experiments on PP, the event rate of getting a correct solution was set to 0.6 for the individual developers and 0.8 for the developer pairs. That is, we expected 33 percent more correct solutions for developer pairs than for individual developers. The estimated total number of experimental subjects required (the sum of individuals and pairs), based on a power of 0.8 as suggested by Cohen [11], [12], was $N = 170$. (If there is a difference between the effect of PP and individual programming, the difference will be revealed in 8 out of 10 replications.)

We also performed a power analysis for a $2 \times 2 \times 3$ fixed-effect analysis of covariance (ANCOVA) with one covariate. The analysis was based on three variables: PP (2 levels), control style (2 levels) and programmer category (3 levels), resulting in 12 levels (or groups). The planned covariate was the pretest of programming skill level. We did not calculate the effect size $f$ from the first phase of this experiment (because the experiment used a different main treatment) but based it on a medium effect size $f = 0.25$ (as suggested by Dybå et al. [16]) for all the three variables and interactions. The power analysis showed that we needed $N = 14$ observations in each of the 12 groups, for a total $N = 168$, to get a minimum power of 0.8 for all *three* main effects and interactions. This is almost identical to the result from the logistic regression-based power analysis ($N = 170$). However, in order to attain a power of 0.8 for the *two* two-level variables, PP and control style, and their interaction, it would suffice to have $N = 72$ observations per level, corresponding to $N = 12$ observations in each group, and a total sample size of $N = 144$.

### 3.2 Population and Sampling Procedure

The target population of the experiment was professional Java consultants. To obtain a broad sample of this population, subjects were hired from a total of 29 software consultancy companies in Norway, Sweden, and the UK. Since students were not part of the target population, the student subjects from the first phase were removed from the final sample, which left a total sample of 295 professional junior, intermediate, and senior Java consultants constituting 197 experimental subjects (99 individuals and 98 pairs). An overview of the subjects' education and experience is given in Appendix A.

To recruit professional developers, several Java consultancy companies were contacted through their formal sales channels. A contract for payment and a time schedule were agreed upon. As in ordinary programming projects, the companies were paid normal consultancy fees for the time spent on the experiment by the consultants (from 5 to 8 hours each, depending on the time spent by the consultant). Seniors were paid more than intermediates, who, in turn, were paid more than juniors. For a few companies, a fixed honorarium (the same payment) for all three developer categories was agreed upon. The participating developers did not receive any information regarding the categorization or actual payment.

A project manager in each company selected the subjects from the company's pool of consultants and rated them according to their Java programming experience (junior, intermediate, and senior), corresponding to how they would rate them for similar kinds of "real" projects. Consequently, a few consultants with ample general work or programming experience (but very little OO or Java experience) could still be rated as "junior" by the companies. In phase two of the experiment, the project manager also provided information about the developers' PP experience (in general and with specific subjects from the same company).

### 3.3 Group Assignment and Pair Constitution

The pairs were formed based on the individuals' programmer category and their PP experience. Only 10 subjects claimed to have PP experience, which constituted five of the 98 pairs. The consultants did not know in advance who their partner would be during the experiment.

Within a given programmer category, each subject (individual or pair) was assigned randomly to one of the two control style treatments. Since the individuals and the pairs participated in this experiment with a three-year time difference, the subjects were assigned to the *pair programming* treatments in a nonrandomized way. The pairs were constituted from two programmers within the same company, with the additional requirement that they should belong to the same programmer category. Thus, if there was an uneven number of developers within a programmer category, the last person in the programmer category was removed from the experiment, and the project manager of the given company was notified that person would not be hired after all.

Fig. 2 shows the distribution of the subjects in the different groups. All groups consisted of more than 14 subjects (i.e., the number required, according to the power analysis) except for the junior pair category (12 subjects in the centralized control style group and 13 in the delegated control style group). The power was, thus, slightly reduced in these two groups when still assuming a medium effect size $f = 0.25$. However, the power analysis also showed that we only needed 12 subjects in each cell for a power of 0.8 for the main effects of PP and control style and their interaction. Thus, the risk of a slightly reduced power for the two junior pair groups was accepted due to the experimental budget and time constraints. In the second phase of the experiment (2004/2005), it was difficult to recruit junior developers, probably because the companies employed few new graduates after the decline in the IT market in 2001/2002.

1. See www.spss.com/samplepower/.

### 3.4 Tasks

The experiment included the programming of six change tasks: a training task, a pretest task (*t1*), and four (incremental) main experimental tasks (*t2*, *t3*, *t4*, and *t5*).

**Individual Training Task:** All the subjects were asked to change a small program so that it could read numbers from the keyboard and print them out in reverse order. The purpose of this task was to familiarize the subjects with the experimental environment.

**Individual Pretest Task (Task 1: ATM):** For the pretest task (*t1*), all the subjects implemented the same change on the same design individually. The initial system (before modifications) consisted of seven classes and 354 lines of code. The change consisted of adding transaction log functionality and printing an account statement for a bank teller machine and was not related to the main experiment tasks. This pretest task provided a common baseline for comparing the programming skill level of the subjects.

**Main Experiment Tasks (Tasks 2-5: Coffee Machine):** These tasks were based on two alternative Java systems that were designed and implemented with a centralized and delegated control design strategy, respectively [47]. In a *centralized control* style, a few large "control classes" coordinate a set of simple classes [47]. Alternatively, in a *delegated control* design, a well-defined set of responsibilities are distributed among a number of classes [47]. The classes play specific roles and occupy well-known positions in the application architecture [47], [48]. The centralized control style consisted of seven classes, the delegated control style of 12. Further details are provided in Appendix D. The two design alternatives were coded using similar coding styles, naming conventions, and amounts of comments. Names of identifiers (e.g., variables and methods) were long and reasonably descriptive. UML sequence diagrams of the main scenario for the two designs were given to help clarify the designs. The tasks consisted of four incremental changes to the coffee machine:

t2. *Implement a coin return button.*
t3. *Introduce bouillon as a new drink choice.*
t4. *Check whether all ingredients are available for the selected drink.*
t5. *Make one's own drink by selecting from the available ingredients.*

**Special Last Task (the Time Sink Task):** The final change task in an experiment needs special attention as a result of potential "ceiling effects." If the last task is included in the analyses, it is difficult to discriminate between the performance of the subjects regarding effort and correctness. Subjects who work fast may spend relatively more time on the last task than they would on the earlier tasks. Similarly, subjects who work slowly may have insufficient time to perform the last task correctly. Consequently, the final change task (*t5*) in this experiment was not included in the analysis. Thus, the analysis of duration and effort is not threatened by whether the subjects actually managed to complete the last task, while at the same time, the presence of the last task helped to put time pressure on the subjects during the experiment. Pilot experiments were conducted to ensure that it would be very likely that all subjects would complete tasks *t1-t4* within a maximum time span of eight hours. All pairs and all but two individuals completed *t1-t4*.

### 3.5 Execution and Practical Considerations

The experiment was conducted incrementally in 27 separate sessions on separate days (for the 29 companies); 10 sessions for the individual programming phase and 17 sessions for the PP phase. The experiment was conducted in the subjects' own offices, where each developer would normally work, or in offices at Simula Research Laboratory. Work at Simula was similar to working at a client's site. In both work environments, the subjects had access to the Internet, printers, libraries, coffee, and so on, as in any other project they might be working on. Each subject also used a Java development tool of their own choice, e.g., JBuilder, Eclipse, IntelliJ, NetBeans, or Emacs and Javac. The researchers decided where the pairs should be located to ensure that they did not disturb each other or listen to each others' conversations during the experiment. To ensure accurate duration and effort data, the subjects were also told to only take breaks *between* the tasks. We also instructed the consultants not to answer telephone calls or talk to colleagues (other than the partner for the pair programmers) during the experiment. During each of the 27 sessions, one or several researchers were present on site at all times to assist in case of technical problems and ensure that the subjects followed the prescribed procedure.

The experiment lasted one day and was divided into four sessions (see Fig. 2). First, the participants were given an introduction to the experiment by the first two authors of this paper. The presentation included general information about practical matters pertaining to the experiment (e.g., time to perform the tasks, breaks, lunch, and how to complete the tasks and the questionnaires). In the second phase of the experiment, the subjects were also introduced to the concept of PP during the presentation, which focused on the active collaboration in PP and involved a short description of the two roles (driver and navigator). The developers were told that they could decide themselves how often and when to switch roles, but they had to try both roles (even if only for five minutes). The main message was that they should focus on making the collaboration as efficient as possible. The information provided regarding PP included no results from prior empirical studies. At the end of the presentation, the subjects were informed of the name of their pair programming partner, in addition to usernames and passwords to the Web-based Simula Experiment Support Environment (SESE) [3]; see below.

After the presentation, the participants answered a prequestionnaire about their education and experience before they started performing the training task and the pretest task (*t1*) individually. After they had finished the pretest task, the subjects started to perform the main experiment tasks (*t2-t5*) individually or in pairs.

To support the logistics of the experiment, the subjects used SESE [3] to answer questionnaires, download code and documents, and to upload task solutions. Except for the Java source code, which contained class, method, and variable names and comments in English, all subjects received the experimental material in their first language (Norwegian, Swedish, or English). Each task consisted of the following steps:

1. Download and unpack a compressed directory containing the Java code to be modified. This step was performed only prior to task *t2* for the coffee machine design change tasks because tasks *t3-t5* were based on the solution of the previous task.
2. Download task descriptions (details are provided in [1]). Each task description contained a test case that each subject used to test the solution.
3. Solve the programming task using the chosen development tool.
4. Pack the modified Java code and upload it to SESE.
5. Complete a task questionnaire (details are provided in [1]).

We wanted the subjects to perform the tasks with satisfactory quality in as short a time as possible because most software engineering jobs impose relatively severe time constraints on the tasks to be performed. However, if the time pressure placed on the participating subjects is too high, the quality of the task solutions may be reduced to the point where it becomes meaningless to use the corresponding task times in subsequent statistical analyses. The challenge is, therefore, to impose realistic time pressure on the subjects. What constitutes the best way to meet this challenge depends, to some extent, on the size, duration, and location of an experiment [36]. In this experiment, we used the following strategy:

- Instead of offering an hourly rate, we offered a "fixed" honorarium based on an estimate that the work would take five hours to complete. We told the subjects that they would be paid for those five hours, regardless of how much time they would actually need. Hence, those subjects who finished early (e.g., in two hours) were still paid for five hours. We employed this strategy to encourage the subjects to finish as quickly as possible and to discourage them from working slowly in order to receive higher payment. However, to maintain motivation, once the five hours had passed, we told those subjects who had not finished that they would be paid for additional hours if they attempted to complete their tasks.
- We introduced a special last, time-sink task (see Section 3.4)
- The subjects were allowed to leave when they finished. Those who did not finish had to leave after eight hours.
- The researchers guaranteed strict confidentiality regarding information about the subjects' performance. In particular, no information would be given to the company or to the individuals themselves about their own performance. Furthermore, the subjects signed a confidentiality agreement where they agreed not to reveal any information about the experiment to their peers.

### 3.6 Variables and Analysis Model

This section defines in more precise terms the variables of the experiment, how data was collected for these variables, and the models for analysis used to test the hypotheses.

#### 3.6.1 Dependent Variables

**Duration:** The elapsed time in minutes to complete change tasks *t2-t4*. Before starting on a task, the subjects wrote down the current time. When they had completed the task, they reported the total time (in minutes) for that task. Nonproductive time between the tasks was not included. For the duration measure to be meaningful, we considered duration only for subjects with correct solutions.

**Effort:** The total change effort in person-minutes to complete change tasks *t2-t4*. The total effort for the pairs was thus the duration for the pair multiplied by two. As for duration, we considered effort only for subjects with correct solutions.

**Correctness:** A binary, functional correctness score with value "1" if all change tasks *t2-t4* were implemented correctly and "0" if at least one of these tasks contained serious logical errors. The change task solutions were reviewed by two independent senior consultants who were not among the experimental subjects and were not informed about the hypotheses of the experiment. To perform the correctness analysis, one of the consultants first developed a tool that automatically unpacked and built the source code corresponding to each task solution. In total, this corresponded to almost 900 different Java programs (one pretest task by 295 individuals and three main experiment tasks by 99 individuals and 98 pairs). Then, each solution was tested using an automated test script. For each test run, the difference between the *expected* output of the test case (this test output was given to the subjects as part of the task specifications) and the *actual* output generated by each program was computed. The tool also showed the complete source code, as well as the differences in source code between each version of the program delivered by each subject, to identify exactly how they had changed the program to complete the change task. To perform the final grading of the task solutions, a Web-based grading tool was developed that enabled the consultants to view the source code, the difference in source code, the expected and actual test case output, and the difference between the two. The score *correct* was given if there were no, or only cosmetic, differences in the test case output and no additional serious logical errors were revealed by manual inspection of the source code; otherwise, the score *incorrect* was given. Each time a mistake was identified, the reason for giving $\mathrm{correct} = 0$ was recorded. One consultant was responsible for phase 1, the other for phase 2, but, essentially, the correctness scores of both phases were based on a consensus between the two

consultants, not two separate "scores": To ensure consistent scoring, the second consultant adopted the same, explicit decision criteria as the first consultant had recorded for the phase 1 solutions. Furthermore, to ensure proper use of the criteria, the second consultant first redid one-third of the evaluations performed by the first consultant as an exercise and discussed those evaluations in the second phase with the first consultant when in doubt. The consultants performed their analysis twice to avoid inconsistencies in the way they had graded the task solutions. Completing this work took approximately 200 person-hours in total, including both phases of the experiment.

#### 3.6.2 Controlled Factors

**Pair Programming (PP):** Whether the subjects worked in pairs (*Pair*) or individually (*Ind*).

**Control Style (CS):** The two alternative Java implementations of the coffee machine; centralized (*CC*) or delegated (*DC*).

**Programmer Category (ProgCat):** *Junior*, *Intermediate*, or *Senior* Java consultants.

#### 3.6.3 Covariates

**Pretest Duration (pre_dur):** *The time taken (in minutes) to complete the pretest task (t1).* The individual pretest result was used as a covariate that modeled the variation in the dependent variables that could be explained by individual skill differences. Such an approach is known as Analysis of Covariance (ANCOVA) and is commonly used to adjust for differences between groups in quasi-experiments [14]. In this experiment, differences could be expected due to the time gap between the two phases. It was necessary to decide which of the two pretests (one for each individual in the pair) should be used in the ANCOVA model, and this we did by flipping a coin to randomly select one of the two pretests of each pair. In this way, the pretest of a randomly selected individual who worked in a pair was compared with the pretest of a randomly selected individual who worked individually on the same experiment tasks. Several alternatives to this random selection procedure were discussed, such as taking the average duration of the two pretests in a pair or running two separate analyses using the "fastest" and "slowest" pretest of each pair. However, ANCOVA presumes that an identical pretest measure is used for all subjects. Using the "fastest," "slowest," or "average" pretest values would result in different pretest measures for the pairs and individuals. For example, using the "fastest" pretest from the pairs would bias the results in that, on average, ANCOVA overcompensates for the skills of the pairs in favor of the individual programmers. Similarly, using the "slowest" pretest from the pairs would bias the results in favor of the pair programmers. Finally, by taking the "average" of the two pretests, the variance of the resulting measure would be much smaller than for the individuals (the intrapair difference might be large for some pairs and small for others, while still having the same average pretest value). Thus, overall, the random selection of a pretest within a pair seemed most appropriate. In addition, the ANCOVA requirement of identical pretest measures prevented us from allowing the PP subjects to solve the pretest task in pairs, which would lead to different pretest measures (individuals versus pairs).

#### 3.6.4 Model Specifications

A generalized linear model (GLM) approach [28] was used to perform an ANCOVA to test the hypotheses specified in Section 2. The *GENMOD* procedure provided in the statistical software package *SAS* was used to fit the models. A justification for the specifications of the model follows.

Since this experiment was a quasi-experiment, the models needed to account for differences between the groups due to a lack of random assignment. The pretest measure *pre_dur* was used to specify ANCOVA models that adjust the observed responses for the effect of the covariate, as recommended in [14]. The covariate is log-transformed to, among other things, reduce the potential negative effect that outliers can have on the model fit.

Furthermore, the time (duration, effort) and correctness data was not normally distributed, which also affected the model specifications. GLM is the preferred approach to analyzing experiments with nonnormal data [49]. In GLMs, one specifies the distribution of the response $y$ and a link function $g$. The link function defines the scale on which the effects of the explanatory variables are assumed to combine additively. The time data was modeled by specifying a *Gamma* distribution and the $log$ link function. The Gamma distribution is suitable for observations that take only positive values and are skewed to the right, which is the case for time data that has zero as a lower limit and no clear upper limit (though it cannot be longer than eight hours in this experiment). Note that an alternative approach would be to simply log-transform the variable by computing the log of each response $log(y)$ as the dependent variable and using a log-linear model to analyze the data on the assumption that $log(y)$ would be approximately normally distributed. However, unlike such an approach, GLM takes advantage of the *natural* distribution of the response $y$, in our case, *Gamma* for the time data. Furthermore, the expected mean $\mu = E(y)$, rather than the response $y$, is transformed to achieve linearity. As elaborated on in [28], [49], these properties of GLM have many theoretical and practical advantages over transformation-based approaches.

The correctness measure is fitted by specifying a *Binomial* distribution and the *logit* link function in the GENMOD procedure. This special case of GLM is also known as a logistic regression model and is a common choice for modeling binary responses.

Given that the underlying assumptions of the models are not violated,[2] the presence of a significant model term corresponds to rejecting the related null hypothesis. The following terms are used to test the hypotheses:

2. An empirical assessment of the model assumptions is provided in Section 5.1.

TABLE 2
Complete Model Specifications

| Model | Response | Distrib. | Link | Model Term | Primary use of model term |
|---|---|---|---|---|---|
| (1) | Duration | Gamma | Log | Log(pre_dur) | Covariate to adjust for individual skill differences |
| | | | | **PP** | **Test $H0_1$ (Duration Main Effect)** |
| | | | | CS | Models the effect of control style on duration |
| | | | | **PPxCS** | **Test $H0_2$** |
| | | | | **Log(pre_dur)xPP** | **Test $H0_3$** |
| | | | | Log(pre_dur)xCS | Test for ANCOVA assumption of homogeneity in slopes |
| | | | | Log(pre_dur)xPPxCS | Test for ANCOVA assumption of homogeneity in slopes |
| (2) | Effort | Gamma | Log | Log(pre_dur) | Covariate to adjust for individual skill differences |
| | | | | **PP** | **Test $H0_4$ (Effort Main Effect)** |
| | | | | CS | Models the effect of control style on effort |
| | | | | **PPxCS** | **Test $H0_5$** |
| | | | | **Log(pre_dur)xPP** | **Test $H0_6$** |
| | | | | Log(pre_dur)xCS | Test for ANCOVA assumption of homogeneity in slopes |
| | | | | Log(pre_dur)xPPxCS | Test for ANCOVA assumption of homogeneity in slopes |
| (3) | Correctness | Binomial | Logit | Log(pre_dur) | Covariate to adjust for individual skill differences |
| | | | | **PP** | **Test $H0_7$ (Correctness Main Effect)** |
| | | | | CS | Models the effect of control style on correctness |
| | | | | **PPxCS** | **Test $H0_8$** |
| | | | | **Log(pre_dur)xPP** | **Test $H0_9$** |
| | | | | Log(pre_dur)xCS | Test for ANCOVA assumption of homogeneity in slopes |
| | | | | Log(pre_dur)xPPxCS | Test for ANCOVA assumption of homogeneity in slopes |

- The *Pair Programming (PP)* variable models the main effect of *pairs* versus *individuals* on *duration*, *effort*, and *correctness* (to test hypotheses $H0_1$, $H0_4$, and $H0_7$).
- The *Control Style (CS)* variable models the main effect of the control-style *DC* versus *CC* on *duration*, *effort*, and *correctness* as an indicator of *system complexity*. The interaction term between *PP* and *CS*, *PPxCS*, models the moderating effect of the control style on the effect of PP (to test hypotheses $H0_2$, $H0_5$, and $H0_8$).
- While the log-transformed covariate *Log(pre_dur)* is primarily needed to adjust for individual skill differences, recall that it may also be viewed as a measure of programmer expertise. Thus, the interaction term between the log-transformed pretest duration and PP, *Log(pre_dur)xPP* models the moderating effect of programmer expertise on PP by using the *Log(pre_dur)* as a measure of expertise (to test hypotheses $H0_3$, $H0_6$, and $H0_9$).
- The *Programmer Category* (*ProgCat*) is not used directly in the models because the PP subjects had, on average, more programming experience than the subjects who participated as individuals (Appendix A). Thus, it is necessary to include a pretest in the model, and the pretest score is correlated with *ProgCat* (e.g., seniors are faster than juniors). This, in turn, means that it is not possible to interpret a model that includes both *Log(pre_dur)* and *ProgCat* as model terms. However, *ProgCat* is still used as a moderating variable of programmer expertise on PP by performing separate analyses for each level (junior, intermediate, and senior) and comparing the differences between the levels (to test hypotheses $H0_3$, $H0_6$, and $H0_9$).

Table 2 specifies the unreduced models, i.e., all possible interaction terms are included. The unreduced models include the covariate *Log(pre_dur)*, the model terms *PP* and *CS*, and all possible interactions between these model terms. If interaction terms involving the covariate (pretest result) were not significant at $\alpha = 0.05$, they were removed in the final (reduced) model. In the model reduction process, insignificant interaction effects (on the basis of the type III adjusted sums of squares) were removed one at a time, starting with the highest-order and least significant interaction, as in backward elimination.

Note that the nine statistical tests were not completely independent. In particular, since $Effort = 2 * Duration$ for the pairs and $Effort = Duration$ for the individuals, $H0_1$ is very much related to $H0_4$. Furthermore, $H0_2$ equals $H0_5$ and $H0_3$ equals $H0_6$ because they investigate interaction effects that do not change by setting $Effort = 2 * Duration$ for the pairs. Thus, from a technical perspective, this experiment has seven distinct hypotheses tests. Still, we believe that it was useful to perform statistical tests of both $Duration$ and $Effort$ because they offer complementary insights on the costs and benefits of pair programming.

### 3.6.5 Effect Size Calculations

Based on the reduced GLM models, we also calculated the adjusted least square means [28] and the difference in adjusted means for the *PP* and *PPxCS* model terms to assess and visualize the effect sizes for the two levels of *PP* (*Ind* and *Pair*) and for the two levels of *PP* for either the centralized (*CC*) or delegated (*DC*) control style. These calculations were performed for all 12 models (duration, effort, and correctness models for the all subjects and for juniors, intermediates, and seniors).

TABLE 3
Unreduced and Final Models Including *All* Developer Categories

| | | Unreduced Model | | | | Reduced (Final) Model | |
|---|---|---|---|---|---|---|---|
| Model | H0 | Model Term | DF | Chi-Square | Pr > ChiSq | Chi-Square | Pr > ChiSq |
| (1) Duration | | Log(pre_dur) | 1 | 29.33 | <.0001 | 31.36 | <.0001 |
| | H01 | PP | 1 | 0.03 | 0.8607 | 1.48 | 0.2239 |
| | | CS | 1 | 2.38 | 0.1231 | 13.11 | 0.0003 |
| | H02 | PPxCS | 1 | 1.48 | 0.2238 | 5.59 | 0.0181 |
| | H03 | Log(pre_dur)xPP | 1 | 0.13 | 0.7210 | | |
| | | Log(pre_dur)xCS | 1 | 1.00 | 0.3178 | | |
| | | Log(pre_dur)xPPxCS | 1 | 0.69 | 0.4077 | | |
| (2) Effort | | Log(pre_dur) | 1 | 29.33 | <.0001 | 31.36 | <.0001 |
| | H04 | PP | 1 | 2.94 | 0.0865 | 62.08 | <.0001 |
| | | CS | 1 | 2.38 | 0.1231 | 13.11 | 0.0003 |
| | H05 | PPxCS | 1 | 1.48 | 0.2238 | 5.59 | 0.0181 |
| | H06 | Log(pre_dur)xPP | 1 | 0.13 | 0.7210 | | |
| | | Log(pre_dur)xCS | 1 | 1.00 | 0.3178 | | |
| | | Log(pre_dur)xPPxCS | 1 | 0.69 | 0.4077 | | |
| (3) Correctness | | Log(pre_dur) | 1 | 11.71 | 0.0006 | 13.32 | 0.0003 |
| | H07 | PP | 1 | 0.07 | 0.7967 | 0.33 | 0.5641 |
| | | CS | 1 | 4.49 | 0.0340 | 5.28 | 0.0216 |
| | H08 | PPxCS | 1 | 0.23 | 0.6306 | 6.20 | 0.0128 |
| | H09 | Log(pre_dur)xPP | 1 | 0.03 | 0.8567 | | |
| | | Log(pre_dur)xCS | 1 | 3.91 | 0.0480 | 4.90 | 0.0269 |
| | | Log(pre_dur)xPPxCS | 1 | 0.02 | 0.8873 | | |

Since the GLM models for the dependent variables *Duration* and *Effort* use the *log* link function, the least square means estimates produced by the GENMOD procedure were first transformed back to the original time scale (in minutes) by taking the exponential of the adjusted least square means estimates. Similarly, the least square means of the *logit*, i.e., $\mu = log(p/(1-p))$, was transformed back to the expected probability that the solution would be correct ($p = exp(\mu)/(exp(\mu)+1)$).

The adjusted least square means estimates can be compared to the unadjusted sample means, the difference being that the former were adjusted for the effect of the covariate on the responses. Finally, to assess the effect size, the contrasts or *differences* in least square means and 95 percent confidence intervals on the differences were computed. For the dependent variables *Duration* and *Effort*, the difference was reported as the relative time differences (as percentages) for *Pair* versus *Ind*, *Pair (CC)* versus *Ind (CC)* and *Pair (DC)* versus *Ind (DC)*. For *Correctness*, the difference was calculated as the relative difference in proportions of subjects with correct solutions as well as *odds-ratios* [28] for *Pair* versus *Ind*, *Pair(CC)* versus *Ind (CC)* and *Pair (DC)* versus *Ind (DC)*.

## 4 Results

This section presents the results of the experiment in terms of descriptive statistics, hypotheses tests, and effect sizes.

### 4.1 Overall Results

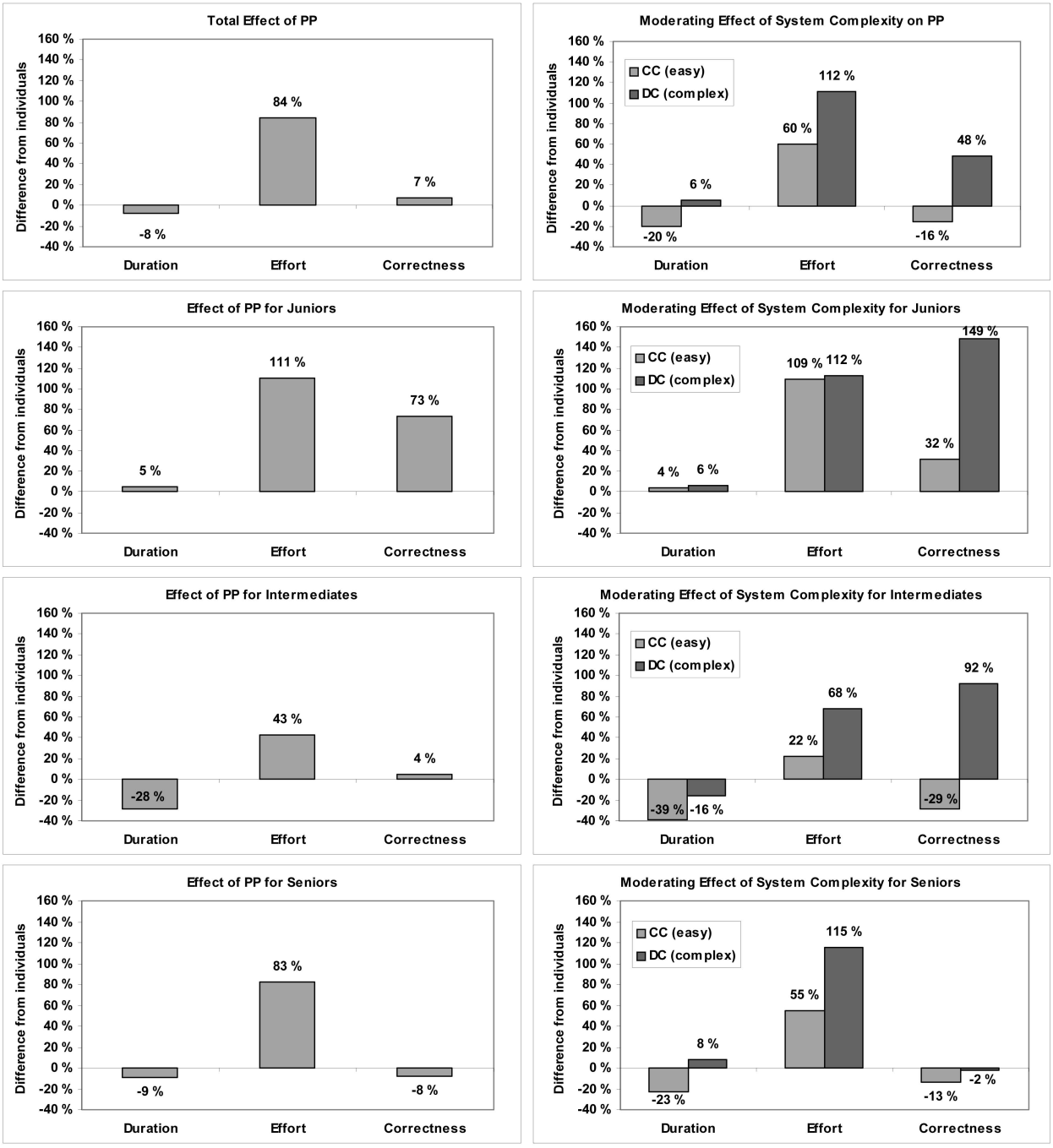
To test the hypotheses, GLM models for the dependent variables *Duration*, *Effort*, and *Correctness* were first fitted on the complete data set, including all categories of developer. Recall that, to provide meaningful measures of duration and effort, only correct solutions were considered in the analyses, with the result that 139 observations (59 individuals and 80 pairs) were used to build the duration and effort models. For correctness, all 197 observations (99 individuals and 98 pairs) were used.

A summary of the three resulting models are given in Table 3, including both the initial, unreduced models (including all interaction terms) and the final models. Due to space constraints, we have not shown the intermediate models, but the results are consistent with those reported in Table 3. In all models, the covariate $Log(pre\_dur)$ was highly significant. This means that the ANCOVA models did adjust the observed responses for the effect of the covariate. More detailed assessments of the underlying ANCOVA and GLM model assumptions are provided in Section 5.1.

Detailed model summaries for the individual programmer categories Junior, Intermediate, and Senior are given in Appendix B.

Appendix C provides the unadjusted and adjusted means and differences in means, respectively, for duration, effort, and correctness of pairs versus individuals, following the procedure described in Section 3.6.5. Also indicated are 95 percent confidence intervals and p-values for all pairwise differences. Fig. 3 visualizes the relative differences of pairs (versus individuals) for the different treatment combinations CS and ProgCat on the basis of the adjusted means provided in Appendix C.

At this point, we will discuss informally the effects of PP on the basis of the differences of pairs versus individuals




(a) (b)

Fig. 3. The effects of pair programming on duration, effort, and correctness for different levels of (a) programmer expertise and (b) system complexity.

shown in Fig. 3. Overall, the pairs had an insignificant 8 percent decrease in the time taken to perform the tasks correctly ($p = 0.2235$), corresponding to an 84 percent increase in effort ($p < 0.0001$). The effect of PP on effort was consistently negative across all treatment groups. There was only a 7 percent increase in the proportion of correct solutions of the pairs compared with the individuals (odds ratio $= 1.28$, $p = 0.5628$).

However, the main effects of PP are masked by the moderating effect of system complexity in the control style (*CS*). More specifically, for the *CC* design, the pairs had a significant 20 percent decrease in duration ($p = 0.0092$), but an insignificant 16 percent decrease in correctness (odds ratio $= 0.46$, $p = 0.2261$). For the *DC* design, the pairs had an insignificant 6 percent *increase* in duration ($p = 0.5472$), but a significant 48 percent increase in correctness (odds ratio $= 3.56$, $p = 0.0194$). Thus, the statistically significant effects of PP are decreased duration on the simpler *CC* design and increased correctness on the more complex *DC* design.

TABLE 4
Results of Hypotheses Tests and Effect Size Estimates

| H0 | Reject? | Effect size estimates |
|---|---|---|
| $H0_1$ | no | The pairs worked 8% faster than did individuals. |
| $H0_2$ | yes | For the *CC* control style, the pairs worked 20% faster than did individuals.<br>For the *DC* control style, the pairs worked 6% slower than did individuals. |
| $H0_3$ | no | Junior pairs worked 5% slower than did junior individuals.<br>Intermediate pairs worked 28% faster than did intermediate individuals.<br>Senior pairs worked 9% faster than did senior individuals. |
| $H0_4$ | yes | The pairs required 84% more effort than did individuals. |
| $H0_5$ | yes | For the *CC* control style, the pairs required 60% more effort than did individuals.<br>For the *DC* control style, the pairs required 112% more effort than did individuals. |
| $H0_6$ | no | Junior pairs required 111% more effort than did junior individuals.<br>Intermediate pairs required 43% more effort than did intermediate individuals.<br>Senior pairs required 83% more effort than did senior individuals. |
| $H0_7$ | no | The pairs had a 7% increase in the proportion of correct solutions (odds ratio = 1.28). |
| $H0_8$ | yes | For the *CC* control style, the pairs had a 16% decrease in the proportion of correct solutions compared with the individuals (odds ratio = 0.46).<br>For the *DC* control style, the pairs had a 48% increase in the proportion of correct solutions compared with the individuals (odds ratio = 3.56). |
| $H0_9$ | no | The junior pairs had a 73% increase in the proportion of correct solutions compared with the individuals (odds ratio = 5.48).<br>The intermediate pairs had a 4% increase in the proportion of correct solutions compared with the individuals (odds ratio = 1.15).<br>The senior pairs had an 8% decrease in the proportion of correct solutions compared with the individuals (odds ratio = 0.65). |

Furthermore, when considering the moderating effect of programmer expertise (*ProgCat*), junior pairs had an insignificant 5 percent increase in duration ($p = 0.7106$) but a significant 73 percent increase in correctness (odds ratio $= 5.46$, $p = 0.0344$). Thus, the juniors benefited from PP in terms of increased correctness. Intermediate pairs had a significant 28 percent decrease in duration ($p = 0.0072$) and an insignificant 4 percent increase in correctness (odds ratio $= 1.15$, $p = 0.8536$). Thus, intermediates also seemed to benefit from PP, but mainly in terms of decreased duration. Senior pairs had an insignificant 9 percent decrease in duration ($p = 0.3015$) and an insignificant 8 percent *decrease* in correctness (odds ratio $= 0.65$, $p = 0.5579$). Thus, there were no significant overall benefits of PP for seniors.

However, when considering the combined moderating effect of system complexity (*CS*) and programmer expertise (*ProgCat*) on PP, there appears to be an interaction effect: Among the different treatment combinations, only juniors assigned to the *DC* design had a significant effect of PP on correctness, with a remarkable 149 percent increase (odds ratio $= 10.48$, $p = 0.0234$) compared with individuals. Furthermore, only intermediates and seniors experienced a significant effect of PP on duration, and that only on the *CC* design, with a 39 percent ($p = 0.0006$) and 23 percent ($p = 0.0255$) decrease, respectively. Note, however, that in this experiment, it was not feasible to test such interactions formally, e.g., by including a model term $PP \times ProgCat \times CS$, since we had to include the covariate *Log(Pre_dur)* to adjust for group differences due to nonrandom assignment to the PP groups. *Log(Pre_dur)* is confounded with *ProgCat* (seniors performed better on the pretest than did the intermediates, who in turn performed better than the juniors). It would thus have been impossible to interpret a model that included both *ProgCat* and *Log(Pre_dur)*. Consequently, we had no choice but to assess this apparent interaction on the basis of the pairwise differences as depicted in Fig. 3.

## 4.2 Formal Tests of Hypotheses

The formal tests of the hypotheses and the related effect size estimates (as specified in Section 3.6) are summarized in Table 4.

### 4.2.1 The Effect of Pair Programming on Duration

***$H0_1$ (The duration to perform change tasks is equal for individuals and pairs):*** The model term PP in model 1 is not significant with $p = 0.2239$. Thus, there is insufficient support for the hypothesis that pair programmers perform change tasks faster than individual programmers.

***$H0_2$ (The difference in the duration to perform change tasks for pairs versus individuals does not depend on system complexity):*** The model term *PP*CS* in model 1 in Table 3 is significant with $p = 0.0181$. Hence, we accept the alternative hypothesis that the effect of PP on duration depends on system complexity.

***$H0_3$ (The difference in the duration to perform change tasks for pairs versus individuals does not depend on the programmer expertise):*** The model term *Log(pre_dur)xPP* in model 1) is not significant with $p = 0.30$. There is insufficient support for the alternative hypothesis that the effect of PP on duration depends on expertise as measured by the pretest duration.

When considering Programmer Category (*ProgCat*) as the moderating variable instead of the pretest *Log(pre_dur)*, $H0_3$ is still not rejected at $\alpha = 0.05$. To see this, we consider the confidence intervals of the adjusted difference in least square means for the model term *PP* (Table 9 in Appendix C): The 95 percent confidence intervals of the

adjusted difference in least square means for the model term *PP* are ($-20$ percent, $+38$ percent), ($-44$ percent, $-9$ percent), and ($-23$ percent, $+9$ percent) for juniors, intermediates, and seniors, respectively. Since they all overlap, $H0_3$ is not rejected when using the moderating variable *ProgCat*.

#### 4.2.2 The Effect of Pair Programming on Effort

***H0$_4$ (The effort spent to perform change tasks is equal for individuals and pairs):*** The model term *PP* in model 2 is significant with $p < 0.0001$. Hence, we accept the alternative hypothesis that pair programmers require more effort to perform change tasks than individual programmers.

***H0$_5$ (The difference in the effort spent to perform change tasks for individuals and pairs does not depend on system complexity):*** The model term *PP*CS* in model 2 in Table 3 is significant with $p = 0.0181$. Hence, we accept the alternative hypothesis that the effect of PP on effort depends on system complexity.

***H0$_6$ (The difference in the effort spent to perform change tasks for pairs versus individuals does not depend on programmer expertise):*** The model term *Log(pre_dur)xPP* in model (2) is not significant with $p = 0.7210$. Hence, there is insufficient support for the alternative hypothesis that the effect of PP on effort depends on expertise as measured by the pretest duration.

When considering Programmer Category (*ProgCat*) as the moderating variable instead of the pretest *Log(pre_dur)*, $H0_6$ is still not rejected at $\alpha = 0.05$: From Table 10 in Appendix C, we see that the 95 percent confidence intervals of the adjusted difference in least square means for the model term PP are ($+61$ percent, $+176$ percent), ($+12$ percent, $+83$ percent), and ($+53$ percent, $+117$ percent) for juniors, intermediates, and seniors, respectively. Since they all overlap, $H0_6$ is not rejected when using the moderating variable *ProgCat*.

#### 4.2.3 The Effect of Pair Programming on Correctness

***H0$_7$ (The correctness of the maintained programs is equal for individuals and pairs):*** The model term *PP* in model 3 is not significant with $p = 0.5641$. There is insufficient support for the hypothesis that pair programmers produce more correct programs than do individuals.

***H0$_8$ (The difference in the correctness of the maintained programs for pairs versus individuals does not depend on system complexity):*** The model term *PP*CS* in model 3 in Table 3 is significant with $p = 0.0128$. Hence, we accept the alternative hypothesis that the effect of PP on correctness depends on system complexity.

***H0$_9$ (The difference in the correctness of the maintained programs for pairs versus individuals does not depend on programmer expertise):*** The model term *Log(pre_dur)xPP* in model 3 is not significant with $p = 0.8567$. Hence, there is insufficient support for the alternative hypothesis that the effect of PP on correctness depends on expertise as measured by the pretest duration.

When considering Programmer Category (*ProgCat*) as the moderating variable instead of the pretest *Log(pre_dur)*, *H0_9* is still not rejected at $\alpha = 0.05$: From Table 11 in Appendix C, we see that the 95 percent confidence intervals of the adjusted odds ratios for *PP (Pair versus Ind)* are $(1.13, 26.33)$, $(0.25, 5.23)$, and $(0.15, 2.75)$ for juniors, intermediates, and seniors, respectively. Since they all overlap, $H0_9$ is not rejected when using the moderating variable *ProgCat*.

## 5 Threats to Validity

The reported experiment is very realistic compared with previously reported experiments on PP. The hypotheses were formulated in such a way that the results obtained could be generalized to a target population of professional Java consultants performing real programming tasks with professional development tools in a realistic work setting. However, this is an ambitious goal; one that is difficult to achieve. For example, there is a trade-off between ensuring realism (to reduce threats to *external* validity) and ensuring control (to reduce threats to *internal* validity). This section discusses what we consider to be the most important threats to the validity of this experiment and offers suggestions for improvements in future experiments.

### 5.1 Validity of Statistical Conclusions

Validity of statistical conclusions concerns 1) whether the presumed cause and effect covary and 2) how strongly they covary. For the first of these inferences, one may incorrectly conclude that cause and effect covary when, in fact, they do not (a Type I error) or incorrectly conclude that they do not covary when, in fact, they do (a Type II error). For the second inference, one may overestimate or underestimate the magnitude of covariation, as well as the degree of confidence that the estimate warrants [34].

The individual level of significance for the hypotheses tests were set to $\alpha = 0.05$. It is difficult to provide arguments for a specific predetermined significance level for tests of the hypotheses. Setting the alpha-level ultimately involves a subjective assessment of the severities of committing Type I versus Type II errors. In this experiment, we performed a power analysis that showed that we needed almost 300 subjects (100 individuals and 100 pairs) to test the hypotheses at a 0.05 level of significance (for individual hypotheses). However, since we performed multiple tests, there was an increased probability (above 0.05) of falsely rejecting one or more of the null hypotheses. In our case, the nine hypotheses were carefully formulated before any analysis was performed, and, as discussed in [24], most statisticians would, in such cases, nevertheless consider it appropriate to test at an individual 0.05 level, that is, without any adjustments of the alpha-level. Some readers might still prefer a stricter and even more conservative interpretation of the results, by adjusting for the multiplicity of tests. In our case, adjusting the significance level using *Holm's procedure* would be more appropriate than using the even more conservative Bonferroni adjustment, since the Bonferroni adjustment ignores the correlation between tests [21]. We performed a post hoc

assessment at the overall level of significance of our hypotheses, adjusted for test multiplicity, using Holm's procedure on the basis of the p-values reported in Table 3 (note again that we had seven distinct tests in this experiment). The results of applying this procedure suggested that the probability of falsely rejecting at least one out of the seven distinct null hypotheses was less than 0.10; that is, the results reported in Table 4 had an overall level of significance equal to $\alpha = 0.10$. We considered this to be an acceptable trade-off between committing Type I and Type II errors.

An important assumption of Analysis of Covariance is that the slope of the covariate can be considered equal across all treatment combinations [14]. For the duration and effort models (Table 3), no interaction terms involving the covariate *Log(pre_dur)* were significant, indicating that the homogeneity in the slopes assumption was not violated. However, for the correctness model, the model term *Log(pre_dur)xCS* was significant ($p = 0.0269$). This term was thus included in the reduced model of correctness. A similar interaction effect was found for the correctness model that considered only intermediates (Table 7, Appendix B). This implies that the difference in correctness of the two control styles (*CC* versus *DC*) varies for different values of *Log(pre_dur)*, and, hence, there is no meaningful overall interpretation of the effect of *CS* on correctness. Fortunately, no interaction terms involving our main model term *PP* and *Log(pre_dur)* were significant, so it does not complicate the interpretation of the hypotheses tests: Since the *odds* ($p/(1-p)$) in a logistic model can be expressed in a multiplicative model [28], e.g.,

$$p/(1-p) = exp(\beta_0) * exp(\beta_1 x_1) * exp(\beta_2 x_2) * \ldots * exp(\beta_n x_n),$$

the effect of the interaction term *Log(pre_dur)xCS* in the correctness model will cancel out in the affected hypotheses tests ($H0_7$ and $H0_8$), which only need to consider the relative *ratio* in odds (expressed as an *odds ratio* [28]) of *Pair* versus *Ind*, *Pair(CC)* versus *Ind(CC)*, and *Pair(DC)* versus *Ind(DC)*, respectively.

The GLM model assumptions were checked by assessing the deviance residuals [28]. For the logistic model, a plot of the deviance residuals indicated no potentially overinfluential observations. We also performed a Hosmer and Lemeshow Goodness-of-Fit test, which did not indicate a lack of fit ($p = 0.64$). For the duration and effort models, the plot of the deviance residuals indicated that two observations could have been treated as outliers; otherwise, there was no indication of model inadequacy. Removing the two observations from the data set had only very minor effects on parameter estimates and p-values. Thus, we do not consider model misspecification a major threat to validity.

Simulations have shown that quasi-experiments with nonequivalent groups usually have less power than randomized block designs with the same number of subjects, despite the ANCOVA adjustments [14]. There is no simple way to estimate power in quasi-experiments [14]. Thus, despite the large number of subjects, the possibility cannot be ruled out that our results suffer from lack of power. In particular, it is possible that the "nonsignificant" results with regard to the hypotheses on interaction effects between PP and programmer category are due to lack of power because, in this case, the observations were partitioned into tree subsets consisting of only juniors, intermediates, and seniors, respectively. This, of course, increases the error term and, hence, the confidence intervals on the parameter estimates on the effect of PP.

Our decision to conduct a quasi-experiment was guided by practical considerations and costs. In particular, we had already performed the first phase of the experiment with individuals [2]. An alternative would have been to perform a completely new, but smaller, randomized block experiment with, say, 50 individuals and 50 pairs. The pretest could, of course, be used in a randomized block design as well. In such a case, the pretest covariate would serve mainly to reduce the error term of the model, resulting in narrower confidence intervals of the parameter estimates, but would not adjust the estimated population means [14]. In summary, there is a nontrivial trade-off between having fewer subjects in a randomized block design or more subjects in a nonequivalent group design. It is not obvious which is better from the point of view of validity.

### 5.2 Internal Validity

The *internal validity* of an experiment is "the validity of inferences about whether observed covariation between *A* (the presumed treatment) and *B* (the presumed outcome) reflects a causal relationship from *A* to *B* as those variables were manipulated or measured" [34]. If changes in B have causes other than the manipulation of *A*, there is a threat to the internal validity.

The main threat to internal validity in this experiment was the lack of random assignment to the two treatment groups: *individual programming* and *pair programming*. The first phase of this experiment was conducted on individual developers. The second phase was conducted on developer pairs three years later. This might lead to differences in skill between the individuals and the pairs. Different subjects also used different development tools and were from different countries with different cultures. Such differences may also bias the results. It was to assess this threat that we included a pretest task in the experiment. The results suggested that the PP group did possess greater skill, as indicated by their spending less time on performing the pretest task. Consequently, the time spent on the pretest was used to adjust for differences between groups in an ANCOVA model, which is a recommended practice in quasi-experiments [14].

Another related issue is that, for the analyses of duration and effort, we removed subjects with incorrect solutions, thereby introducing a potential bias, particularly since we removed a larger proportion of individuals than pairs. Following the same arguments as above, the inclusion of the pretest in the ANCOVA models will adjust for skill differences, even if the differences were caused by removing subjects with incorrect solutions.

However, for the ANCOVA approach to be effective, the pretest must reflect the expected performance on the main tasks of the experiment, but it is unlikely that it is a "perfect" predictor. For example, and for technical reasons explained in Section 3.6.3, we had to choose only one out of two pretests (at random) to reflect the skills of the two individuals subsequently formed into pairs, and that pretest may not adequately reflect the expected performance of the pairs. Still, in our case, the pretest was consistently a very good predictor of all three dependent variables, and, as seen in Appendix C, it resulted in considerable adjustments in the mean square estimates. We also considered a number of other candidate covariates, such as the *correctness* of the pretest, number of lines of code written in various programming languages, developer tool, country, and programmer category. None of these alternative covariates were significant when the *Log(pre_dur)* term was included in the models, which indicates that the pretest duration was the best measure among those available to us, and the only measure needed to perform the necessary adjustments.

## 5.3 Construct Validity

Construct validity concerns the degree to which inferences are warranted, from 1) the observed persons, settings, and cause and effect operations included in a study to 2) the constructs that these instances might represent. The question, therefore, is whether the sampling particulars of a study can be defended as measures of general constructs [34].

### 5.3.1 Pair Programming

PP has many facets regarding how the pairs are constituted, which tasks are performed, and how the pairs work. Our PP construct is limited in scope to situations where "homogenous" pairs (pairs of programmers with similar competence levels, as indicated by the programmer categorization provided by their common project manager) perform small maintenance tasks on systems of which they had no prior knowledge. The rationale for homogenous pairs was that previous studies on PP have reported that pairs consisting of individuals with similar competence levels in the programming language used (e.g., high-high and low-low) collaborated more successfully than did pairs consisting of individuals with different competence levels in the programming language used (e.g., low-high and low-medium) [9], [19], [43]. Using "heterogeneous" pairs might have been more suitable if the goal had been to study the effect of PP on training and knowledge transfer [19], [26], [42].

A majority of the subjects had little or no experience with PP before the experiment and, in most cases, had not pair programmed with their assigned partners before. Consequently, the results of this study might be a quite conservative measure of the effects of PP, since the pairs had probably not reached their maximum level of combined efficiency during the experiment: Anecdotal evidence suggests that it takes developers from a few hours to a few days to make the transition from individual programming to efficient pair programming [43]. Let this concept of a pair having reached its maximum level of efficiency henceforth be designated "pair jelling."

To address these threats to construct validity, future experiments should consider using heterogeneous pairs and programmers that have already reached their maximum level of combined efficiency.

### 5.3.2 Programmer Expertise

In this experiment, the concept programmer expertise was operationally defined by two variables: 1) the pretest duration in *Log(minutes)* to solve an individual pretest task and 2) the categorization of developers as *Junior*, *Intermediate*, and *Senior* consultants. The pretest measure is a quite direct measure of programmer expertise but clearly does not cover all aspects of expertise that can influence the performance of programmers. However, the extent to which it affects their performance on the particular set of experimental tasks was assessed by including the pretest performance as a covariate in the statistical models. It was consistently a very good predictor of their performance on the subsequent experimental tasks. The developer categories reflect a more overall measure of programmer expertise in the sense that *senior* programmers in a company are considered to have greater programmer expertise than *junior* or *intermediate* programmers, although someone who is considered as an intermediate consultant in one company might be considered as a senior in another company. Hence, the reliability of the categorization is questionable. Nevertheless, as is evident from Appendix C, seniors worked faster (with more correct task solutions) than did intermediates, who, in turn, worked faster (with more correct task solutions) than did juniors.

### 5.3.3 System Complexity

An important threat to construct validity is the extent to which the actual Java systems represent the concept studied, that is, a *simple* system on the basis of having to perform changes on a program with a centralized control style and a *complex* system on the basis of having to perform changes on a program with a delegated control style. A complication is that system complexity probably cannot be expressed in absolute terms because it may depend on the skill of programmers, development processes, etc. From the previous study with individuals, it was clear that juniors found the delegated control style more difficult to change than seniors did. Although the difference in complexity of the two control styles was sufficiently large to demonstrate a moderating effect for system complexity, the effects might have been greater if the systems had had a greater difference in complexity.

### 5.3.4 Correctness

The concept of correctness was operationally defined by the binary dependent variable *Correctness*, which indicated whether the subjects produced functionally correct solutions on *all* the change tasks (t2-t4), thus producing a working final program. As described in Section 3.6.1, a

significant amount of effort was expended on ensuring that the correctness scores were valid. Of course, correctness can be measured in many ways, e.g., number of passed test cases, correctness per task or aggregated across tasks, and number of correctly implemented tasks. It is possible that the results would have been different had we used more fine-grained measures of correctness, but such measures would also complicate the analyses even further. Furthermore, functional correctness represents only one dimension of the more general concept of *quality*, but we do consider functional correctness to be, at least, a very important dimension of quality, and the first candidate as a dependent variable. In a sense, if a program does not deliver the required functionality, it may not be meaningful to consider other aspects of quality. Functional correctness is also more objective than, say, extensibility.

### 5.4 External Validity

The issue of external validity concerns whether a causal relationship holds 1) over variations in persons, settings, treatments, and outcomes that were in the experiment and 2) for persons, settings, treatments, and outcomes that were not in the experiment [34].

Clearly, the experimental systems in this experiment were very small compared with industrial object-oriented software systems. Furthermore, the change tasks were also relatively small in size and duration. However, the change task questionnaires received from the participants after they had completed the change tasks indicate that the *complexity* of the tasks was quite high. Note also that change tasks can be small in industry as well. Nevertheless, we cannot rule out the possibility that the observed effects would have been different if the systems and tasks had been larger.

The scope of this study is limited to situations in which the developers have no prior knowledge of the system to be changed. It is possible that the results do not apply to situations in which the developers are also the original designers. A related issue is whether the short-term effects observed in this experiment are representative of long-term development, in particular, due to pair jelling, as already discussed in Section 5.3.1.

## 6 Discussion

Anecdotal and empirical evidence reported in the literature suggest several organizational and personal benefits of PP over individual programming, such as reduced time to market [10], [25], [26], [32], [44], [46], reduced development costs [10], [20], [25], [26], [33], [44], [46], improved quality of the software [10], [17], [27], [30], [32], [44], [46], reduced costs of training new personnel [43], and enhanced trust, motivation, and information and knowledge transfer among developers [5], [10], [13], [17], [27], [29], [41], [42], [44], [45], [46]. However, in what follows, we focus on comparing and discussing the results of our experiment in relation to other, *controlled experiments* that have assessed the effect of PP on duration, effort, and correctness. In addition, we draw out some implications of these results for both research and practice.

### 6.1 Duration

In previous experiments, the differences in the time taken to perform programming tasks varied from no difference [31], an insignificant (at $\alpha = 0.05$) 29 percent decrease [32], a 42.5 percent decrease [44], [46], a 46.6 percent decrease [30], and a 52 percent decrease in favor of PP [25]. Note that, except for the study reported in [32], the differences in duration were not statistically tested. In contrast, the results reported in this paper suggest an overall insignificant decrease in duration of 8 percent. When considering the moderating effects of system complexity, the results suggest an overall significant 20 percent decrease in duration in favor of PP for the CC design and an insignificant increase of 6 percent in favor of individual programming for the DC design. When also accounting for different levels of programmer expertise, the difference in duration ranged from a decrease of up to 39 percent in favor of PP (for intermediates on the CC design) to a slight increase of 8 percent in favor of individual programming (for seniors on the DC design).

### 6.2 Effort

In this context, effort is simply the same as the duration for the individuals and the duration multiplied by two for the pairs. The differences in the existing studies range from doubled effort [31], a 42 percent increase [32], an insignificant 15 percent increase [44], [46], an insignificant 7 percent increase (13 percent excluding rework) [30], and a 4.2 percent decrease [25]. Note that, except for the studies reported in [44], [46], [30], the differences in effort were not statistically tested. In contrast, the results reported in this paper suggest an overall significant 84 percent increase in effort. When considering the moderating effects of both system complexity and programmer expertise, the difference in effort ranged from an insignificant 22 percent increase (for intermediates on the CC design) to a significant 115 percent increase (for seniors on the DC design), both in favor of individual programming.

### 6.3 Correctness

The results vary from apparently no increase in correctness when using PP (measured in terms of number of resubmissions required to produce a correct program) [31], a significant 15 percent increase in program correctness [44], [46], an insignificant 29 percent increase [30], and a significant 33 percent increase [32]. Note that [31] did not test for significance and [25] did not report any correctness measure. In contrast, the results reported in this paper suggest an overall 7 percent insignificant increase in correctness (72 percent and 76 percent correct solutions for, respectively, individuals and pairs). When considering the moderating effects of system complexity and expertise, our results suggest a significant, overall 48 percent increase in correctness for the DC design (55 percent and 81 percent correct solutions for, respectively, individuals and pairs), but this effect was mostly due to the observations for junior pair programmers, who had a 149 percent increase in

TABLE 5
Descriptive Statistics of the Subjects' General Education, Computer Science (CS) Education, Work Experience, and Programming Experience (in Years)

| Variable | Treatment | Block | N | Mean | Median | StDev | Min | Max |
|---|---|---|---|---|---|---|---|---|
| Education (years) | Individual | Junior | 31 | 4.1 | 4.0 | 1.2 | 0.3 | 6.3 |
| | | Intermediate | 32 | 4.2 | 4.1 | 1.7 | 0.1 | 10.0 |
| | | Senior | 36 | 4.0 | 4.0 | 2.4 | 0.0 | 14.0 |
| | | Total | 99 | 4.1 | 4.0 | 1.9 | 0.0 | 14.0 |
| | Pair | Junior | 50 | 4.5 | 4.5 | 1.3 | 0.0 | 8.0 |
| | | Intermediate | 70 | 4.1 | 4.3 | 1.6 | 0.0 | 8.0 |
| | | Senior | 76 | 3.9 | 4.0 | 1.7 | 0.0 | 7.0 |
| | | Total | 196 | 4.1 | 4.3 | 1.5 | 0.0 | 8.0 |
| CS Education (years) | Individual | Junior | 31 | 1.3 | 1.0 | 1.0 | 0.1 | 4.0 |
| | | Intermediate | 32 | 1.5 | 1.4 | 1.0 | 0.0 | 3.5 |
| | | Senior | 36 | 1.8 | 1.5 | 1.1 | 0.0 | 4.0 |
| | | Total | 99 | 1.5 | 1.5 | 1.1 | 0.0 | 4.0 |
| | Pair | Junior | 50 | 2.3 | 2.3 | 1.5 | 0.0 | 6.0 |
| | | Intermediate | 70 | 1.9 | 2.0 | 1.4 | 0.0 | 5.0 |
| | | Senior | 76 | 2.1 | 1.8 | 1.7 | 0.0 | 7.0 |
| | | Total | 196 | 2.1 | 2.0 | 1.5 | 0.0 | 7.0 |
| Work Exp. (years) | Individual | Junior | 31 | 2.9 | 1.0 | 4.8 | 0.0 | 27.0 |
| | | Intermediate | 32 | 5.8 | 3.0 | 7.7 | 0.0 | 35.0 |
| | | Senior | 36 | 7.6 | 6.5 | 5.5 | 0.0 | 27.0 |
| | | Total | 99 | 5.5 | 4.0 | 6.4 | 0.0 | 35.0 |
| | Pair | Junior | 50 | 5.1 | 3.0 | 6.7 | 0.0 | 35.0 |
| | | Intermediate | 70 | 9.9 | 7.8 | 6.9 | 0.0 | 28.0 |
| | | Senior | 76 | 12.4 | 11.5 | 6.6 | 3.0 | 30.0 |
| | | Total | 196 | 9.7 | 7.8 | 7.3 | 0.0 | 35.0 |
| Programming Exp. (years) | Individual | Junior | 30 | 1.5 | 1.0 | 4.1 | 0.0 | 23.0 |
| | | Intermediate | 32 | 4.5 | 2.0 | 6.7 | 0.0 | 26.0 |
| | | Senior | 36 | 6.3 | 5.0 | 5.4 | 0.0 | 27.0 |
| | | Total | 98 | 4.3 | 2.0 | 5.8 | 0.0 | 27.0 |
| | Pair | Junior | 50 | 3.2 | 1.0 | 5.4 | 0.0 | 25.0 |
| | | Intermediate | 70 | 8.0 | 6.0 | 6.3 | 0.0 | 24.0 |
| | | Senior | 76 | 11.2 | 10.0 | 6.4 | 1.0 | 30.0 |
| | | Total | 196 | 8.0 | 6.0 | 6.7 | 0.0 | 30.0 |

correctness for the DC design (34 percent and 84 percent correct solutions for, respectively, individuals and pairs).

### 6.4 Potential Explanations for the Different Results

There are several differences between the studies that complicate the above attempt to directly compare the results. For example, our duration and effort measures are not directly comparable to those reported in [32] and [44] because we only considered time and effort data for subjects with correct programs, whereas [32] and [44] considered duration and effort regardless of program correctness. Furthermore, in [30], the time spent on a peer review was also included, so an exact comparison of the duration and effort data is difficult.

There are also important differences in the study settings. For example, to properly account for pair jelling, the first assignment was excluded when assessing the benefits of PP in [46], and, as discussed in Section 5.3.1, it is possible that the tasks in other experiments (including ours) were too short for pair jelling to have an optimal, positive effect.

The size and complexity of the programming tasks were also different, and even more importantly, the previous experiments all considered *initial* development tasks, whereas we considered maintenance tasks on systems of which the programmers had no prior knowledge. Note also that in one of the experiments [25], the tasks were not programming tasks, but multiple choice "deduction problems" on procedural algorithms. Hence, it is unclear how the results may apply to PP.

Finally, the subject sample sizes and sample populations differed among the experiments. The difference in power and in the subjects' ability, education, experience, training, etc., in general, and in PP in particular, may be a major cause of the different results.

### 6.5 Implications for Research and Practice

The existing body of empirical evidence indicates that PP affects duration, effort, and correctness, and we are reasonably sure that these effects are not simply due to chance. Thus, we believe that the existing results constitute necessary and useful steps toward being able to predict

TABLE 6
GLM Models for Juniors

| | | Unreduced Model | | | | Reduced (Final) Model | |
|---|---|---|---|---|---|---|---|
| Model | H0 | Model Term | DF | Chi-Square | Pr > ChiSq | Chi-Square | Pr > ChiSq |
| (1) Duration | | Log(pre_dur) | 1 | 1.17 | 0.2794 | 4.25 | 0.0393 |
| | H03 | PP | 1 | 0.41 | 0.5204 | 0.14 | 0.7109 |
| | | CS | 1 | 0.00 | 0.9724 | 1.56 | 0.2122 |
| | | PPxCS | 1 | 0.14 | 0.7083 | 0.01 | 0.9360 |
| | | Log(pre_dur)xPP | 1 | 0.45 | 0.5046 | | |
| | | Log(pre_dur)xCS | 1 | 0.02 | 0.9010 | | |
| | | Log(pre_dur)xPPxCS | 1 | 0.14 | 0.7108 | | |
| (2) Effort | | Log(pre_dur) | 1 | 1.17 | 0.2794 | 4.25 | 0.0393 |
| | H06 | PP | 1 | 0.00 | 0.9806 | 21.14 | <.0001 |
| | | CS | 1 | 0.00 | 0.9724 | 1.56 | 0.2122 |
| | | PPxCS | 1 | 0.14 | 0.7083 | 0.01 | 0.9360 |
| | | Log(pre_dur)xPP | 1 | 0.45 | 0.5046 | | |
| | | Log(pre_dur)xCS | 1 | 0.02 | 0.9010 | | |
| | | Log(pre_dur)xPPxCS | 1 | 0.14 | 0.7108 | | |
| (3) Correctness | | Log(pre_dur) | 1 | 0.04 | 0.8512 | 0.01 | 0.9119 |
| | H09 | PP | 1 | 0.44 | 0.5055 | 5.03 | 0.0249 |
| | | CS | 1 | 1.22 | 0.2702 | 0.67 | 0.4116 |
| | | PPxCS | 1 | 0.46 | 0.4991 | 0.96 | 0.3274 |
| | | Log(pre_dur)xPP | 1 | 0.12 | 0.7288 | | |
| | | Log(pre_dur)xCS | 1 | 0.90 | 0.3441 | | |
| | | Log(pre_dur)xPPxCS | 1 | 0.21 | 0.6447 | | |

*when* PP might be beneficial. For example, our results warrant the prediction that PP is mainly useful for junior programmers when solving maintenance tasks that they would perceive to be too complex if they attempted to solve them individually. In this case, junior pair programmers might be able to achieve approximately the same level of correctness in about the same amount of time as senior individuals. Since seniors are more costly (and probably also in shorter supply) than juniors, the effort overhead incurred by pairing two juniors might be acceptable. By performing more experiments with different systems, tasks, and developers, we might eventually be able to provide a fairly comprehensive set of such guidelines for when PP might be beneficial.

However, we are still far from being able to explain *why* we observe the given effects. One area of research that might provide the necessary scientific foundation is social psychology [6], [7], [18], [23], [50]. For example, *social facilitation*, i.e., improvements in performance due to the presence of others, is said to increase on tasks that require simple repetitive or fast responses (simple tasks) but disappear with activities that need more careful or thoughtful actions (complex tasks) [50]. We may speculate that, for the intermediate and senior pairs, the change tasks on the CC design were sufficiently simple to significantly reduce the duration by means of coactive, social facilitation.

Results from social psychology also suggest that, when, including more complex and involving tasks, *social laboring* is possible, i.e., increased performance in a group compared with an individual performance baseline [7]. For the junior pairs, we may speculate that the DC system was sufficiently complex to facilitate collective social laboring. For the intermediate and senior developers, the systems used in this experiment (both CC and DC) might have been too simple to benefit (in terms of correctness) from the effects of social laboring. The above attempt at explaining parts of our results on the basis of research in social psychology is superficial at best and much more research is required. For example, complicating the above discussion is the impact of pair jelling [46], which has its counterpart in more general models of the possible stages of group development that might be required before a group becomes supportive of task performance [38]. The construction of theories on the effects of PP would certainly benefit from collaboration with researchers from other research disciplines, such as social psychology and group dynamics.

## 7 Conclusions and Further Work

A quasi-experiment was conducted to evaluate the effects of PP on the duration of and effort expended on maintenance tasks performed on Java code, and the correctness of the task solutions. The experimental subjects constituted almost 300 professional Java consultants from many companies in several countries, who used their usual tool environment. Compared with previous experiments on PP, the use of such a body of consultants made the experimental setting more realistic. Furthermore, unlike previous experiments, the effect of PP was investigated in the context of performing maintenance tasks. Another unique aspect was that this experiment included the first ever assessment of the moderating effects of system complexity and programmer expertise. The results show that the effects of PP depend on a combination of system complexity and the expertise of the subjects. To a certain degree, these moderating factors may explain the differences in results of previous experiments on the effect of PP on duration, effort, and correctness.

TABLE 7
GLM Models for Intermediates

| | | Unreduced Model | | | | Reduced (Final) Model | |
|---|---|---|---|---|---|---|---|
| Model | H0 | Model Term | DF | Chi-Square | Pr > ChiSq | Chi-Square | Pr > ChiSq |
| (1) Duration | | Log(pre_dur) | 1 | 4.41 | 0.0358 | 3.55 | 0.0597 |
| | H03 | PP | 1 | 0.80 | 0.3711 | 6.84 | 0.0089 |
| | | CS | 1 | 1.94 | 0.1635 | 2.76 | 0.0966 |
| | | PPxCS | 1 | 0.02 | 0.8772 | 2.48 | 0.1155 |
| | | Log(pre_dur)xPP | 1 | 0.37 | 0.5428 | | |
| | | Log(pre_dur)xCS | 1 | 1.59 | 0.2070 | | |
| | | Log(pre_dur)xPPxCS | 1 | 0.14 | 0.7087 | | |
| (2) Effort | | Log(pre_dur) | 1 | 4.41 | 0.0358 | 3.55 | 0.0597 |
| | H06 | PP | 1 | 0.07 | 0.7916 | 7.46 | 0.0063 |
| | | CS | 1 | 1.94 | 0.1635 | 2.76 | 0.0966 |
| | | PPxCS | 1 | 0.02 | 0.8772 | 2.48 | 0.1155 |
| | | Log(pre_dur)xPP | 1 | 0.37 | 0.5428 | | |
| | | Log(pre_dur)xCS | 1 | 1.59 | 0.2070 | | |
| | | Log(pre_dur)xPPxCS | 1 | 0.14 | 0.7087 | | |
| (3) Correctness | | Log(pre_dur) | 1 | 3.23 | 0.0721 | 6.00 | 0.0143 |
| | H09 | PP | 1 | 0.00 | 0.9956 | 0.03 | 0.854 |
| | | CS | 1 | 5.36 | 0.0206 | 5.17 | 0.023 |
| | | PPxCS | 1 | 0.26 | 0.6077 | 4.50 | 0.0338 |
| | | Log(pre_dur)xPP | 1 | 0.00 | 0.9679 | | |
| | | Log(pre_dur)xCS | 1 | 5.02 | 0.0250 | 4.92 | 0.0266 |
| | | Log(pre_dur)xPPxCS | 1 | 0.54 | 0.4640 | | |

The results suggest several practical ways to increase the benefits of PP. In particular, the experiment showed that junior individuals may lack the necessary skills to perform maintenance tasks with acceptable quality, in particular, on more complex systems. Junior pair programmers achieved a significant increase in correctness compared with the individuals and achieved approximately the same degree of correctness as senior individuals. Maintenance is often viewed as requiring less skill than initial system development and is thus often allocated to the more junior staff. Our results suggest that, if juniors are assigned to complex maintenance tasks, they should perform the tasks in pairs.

Future studies on PP should extend the scope of the present study in two important ways. First, our results suggest that the benefits of PP depend on system complexity. Still, our experimental tasks were relatively small and simple, and our results might therefore represent a conservative estimate of the benefits of PP. Future experi-

TABLE 8
GLM Models for Seniors

| | | Unreduced Model | | | | Reduced (Final) Model | |
|---|---|---|---|---|---|---|---|
| Model | H0 | Model Term | DF | Chi-Square | Pr > ChiSq | Chi-Square | Pr > ChiSq |
| (1) Duration | | Log(pre_dur) | 1 | 9.22 | 0.0024 | 20.99 | <.0001 |
| | H03 | PP | 1 | 2.75 | 0.0971 | 1.07 | 0.3020 |
| | | CS | 1 | 0.39 | 0.5307 | 6.92 | 0.0085 |
| | | PPxCS | 1 | 2.48 | 0.1152 | 4.74 | 0.0294 |
| | | Log(pre_dur)xPP | 1 | 3.39 | 0.0656 | | |
| | | Log(pre_dur)xCS | 1 | 1.09 | 0.2975 | | |
| | | Log(pre_dur)xPPxCS | 1 | 1.90 | 0.1682 | | |
| (2) Effort | | Log(pre_dur) | 1 | 9.22 | 0.0024 | 20.99 | <.0001 |
| | H06 | PP | 1 | 7.26 | 0.0071 | 32.79 | <.0001 |
| | | CS | 1 | 0.39 | 0.5307 | 6.92 | 0.0085 |
| | | PPxCS | 1 | 2.48 | 0.1152 | 4.74 | 0.0294 |
| | | Log(pre_dur)xPP | 1 | 3.39 | 0.0656 | | |
| | | Log(pre_dur)xCS | 1 | 1.09 | 0.2975 | | |
| | | Log(pre_dur)xPPxCS | 1 | 1.90 | 0.1682 | | |
| (3) Correctness | | Log(pre_dur) | 1 | 8.39 | 0.0038 | 4.74 | 0.0295 |
| | H09 | PP | 1 | 1.60 | 0.2061 | 0.35 | 0.5532 |
| | | CS | 1 | 0.03 | 0.8540 | 0.03 | 0.8616 |
| | | PPxCS | 1 | 3.14 | 0.0766 | 0.30 | 0.5833 |
| | | Log(pre_dur)xPP | 1 | 2.12 | 0.1458 | | |
| | | Log(pre_dur)xCS | 1 | 0.10 | 0.7471 | | |
| | | Log(pre_dur)xPPxCS | 1 | 2.87 | 0.0903 | | |

TABLE 9
Duration: Unadjusted and Adjusted Means and Differences in Means

| Scope | | | Unadjusted Means | | | Adjusted GLM Least Square Means | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Prog. Cat | Term | CS | Ind | Pair | % diff | Ind | Pair | % diff | Lower 95% CL | Upper 95% CL | P-value |
| All | PP | CC+DC | 87 | 63 | -28 % | 73 | 67 | -8 % | -19 % | 5 % | 0.2235 |
| | PPxCS | CC | 98 | 64 | -35 % | 88 | 70 | -20 % | -32 % | -5 % | 0.0092 |
| | | DC | 72 | 62 | -15 % | 61 | 65 | 6 % | -12 % | 27 % | 0.5471 |
| Junior | PP | CC+DC | 92 | 81 | -12 % | 82 | 86 | 5 % | -20 % | 38 % | 0.7106 |
| | PPxCS | CC | 95 | 85 | -10 % | 88 | 92 | 4 % | -24 % | 44 % | 0.7985 |
| | | DC | 86 | 78 | -9 % | 75 | 80 | 6 % | -28 % | 56 % | 0.7566 |
| Intermed | PP | CC+DC | 104 | 61 | -41 % | 89 | 64 | -28 % | -44 % | -9 % | 0.0072 |
| | PPxCS | CC | 113 | 60 | -47 % | 106 | 65 | -39 % | -54 % | -19 % | 0.0006 |
| | | DC | 87 | 63 | -27 % | 75 | 63 | -16 % | -40 % | 18 % | 0.3137 |
| Senior | PP | CC+DC | 74 | 51 | -31 % | 62 | 57 | -9 % | -23 % | 9 % | 0.3015 |
| | PPxCS | CC | 87 | 53 | -39 % | 75 | 58 | -23 % | -38 % | -3 % | 0.0255 |
| | | DC | 61 | 49 | -20 % | 52 | 56 | 8 % | -14 % | 35 % | 0.5229 |

TABLE 10
Effort: Unadjusted and Adjusted Means and Differences in Means

| Scope | | | Unadjusted Means | | | Adjusted GLM Least Square Means | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Prog. Cat | Term | CS | Ind | Pair | % diff | Ind | Pair | % diff | Lower 95% CL | Upper 95% CL | P-value |
| All | PP | CC+DC | 87 | 125 | 44 % | 73 | 135 | 84 % | 61 % | 110 % | <.0001 |
| | PPxCS | CC | 98 | 127 | 30 % | 88 | 140 | 60 % | 35 % | 89 % | <.0001 |
| | | DC | 72 | 123 | 70 % | 61 | 129 | 112 % | 76 % | 154 % | <.0001 |
| Junior | PP | CC+DC | 92 | 163 | 77 % | 82 | 172 | 111 % | 61 % | 176 % | <.0001 |
| | PPxCS | CC | 95 | 171 | 79 % | 88 | 184 | 109 % | 51 % | 188 % | <.0001 |
| | | DC | 86 | 155 | 82 % | 75 | 160 | 112 % | 45 % | 212 % | 0.0001 |
| Intermed | PP | CC+DC | 104 | 122 | 18 % | 89 | 128 | 43 % | 12 % | 83 % | 0.0040 |
| | PPxCS | CC | 113 | 119 | 6 % | 106 | 130 | 22 % | -8 % | 62 % | 0.1699 |
| | | DC | 87 | 127 | 45 % | 75 | 126 | 68 % | 20 % | 136 % | 0.0028 |
| Senior | PP | CC+DC | 74 | 102 | 39 % | 62 | 114 | 83 % | 53 % | 117 % | <.0001 |
| | PPxCS | CC | 87 | 107 | 23 % | 75 | 116 | 55 % | 23 % | 94 % | 0.0002 |
| | | DC | 61 | 98 | 60 % | 52 | 112 | 115 % | 71 % | 171 % | <.0001 |

ments should, ideally, include larger systems and more complex tasks.

Second, most of our subjects had no experience in PP with their assigned partner. Thus, our results are conservative in the sense that our experimental setting did not allow the potential effects of pair jelling to reach maturity (a prerequisite for what is known as social laboring in social psychology). To assess the pair jelling effect, future experiments should thus include a mix of pairs with different degrees of pair cohesiveness.

## Appendix A

### Descriptive Statistics of Subjects

See Table 5.

## Appendix B

### Detailed Model Summaries for Junior, Intermediate, and Senior Consultants

See Table 6, Table 7, and Table 8.

## Appendix C

### Effect Size Estimates, Including 95 Percent Confidence Intervals

See Table 9, Table 10, and Table 11.

## Appendix D

### Structural Attributes of the Design Alternatives

Table 12 shows the values of coupling (*OMMIC_N*, *OMMIC_L*, and *OMMEC*) and size (*MC* and *CS*) for the two designs. The CC design has larger classes and fewer methods per class than the DC design. At the *system* level, however, Table 12 (the "Sum" column) shows that the overall nonlibrary coupling is about identical, whereas the coupling to library classes, the total number of methods, and the total system size are larger for the DC design. Given these numbers, it is perhaps not evident why many developers (in particular inexperienced developers) find the DC design more "complex" than the CC design, as clearly demonstrated by empirical results [2]. A plausible

TABLE 11
Correctness: Unadjusted Adjusted Means and Differences in Means

| Scope | | | Unadjusted Means | | | Adjusted GLM Least Square Means | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Prog. Cat | Term | CS | Ind | Pair | Odds ratio | Ind | Pair | Odds ratio | Lower 95% CL | Upper 95% CL | P-value |
| All | PP | CC+DC | 60 % | 82 % | 3.01 | 72 % | 76 % | 1.28 | 0.56 | 2.92 | 0.5628 |
| | PPxCS | CC | 68 % | 80 % | 1.93 | 84 % | 71 % | 0.46 | 0.13 | 1.62 | 0.2261 |
| | | DC | 51 % | 83 % | 4.68 | 55 % | 81 % | 3.56 | 1.23 | 10.34 | 0.0194 |
| Junior | PP | CC+DC | 48 % | 84 % | 5.60 | 48 % | 84 % | 5.46 | 1.13 | 26.33 | 0.0344 |
| | PPxCS | CC | 63 % | 83 % | 3.00 | 63 % | 83 % | 2.85 | 0.37 | 22.11 | 0.3175 |
| | | DC | 33 % | 85 % | 11.00 | 34 % | 84 % | 10.48 | 1.38 | 79.91 | 0.0234 |
| Intermed | PP | CC+DC | 53 % | 80 % | 3.53 | 69 % | 72 % | 1.15 | 0.25 | 5.23 | 0.8536 |
| | PPxCS | CC | 65 % | 80 % | 2.18 | 87 % | 62 % | 0.24 | 0.03 | 2.30 | 0.2161 |
| | | DC | 40 % | 80 % | 6.00 | 41 % | 80 % | 5.51 | 0.73 | 41.50 | 0.0975 |
| Senior | PP | CC+DC | 75 % | 82 % | 1.48 | 84 % | 77 % | 0.65 | 0.15 | 2.75 | 0.5579 |
| | PPxCS | CC | 76 % | 79 % | 1.15 | 86 % | 75 % | 0.47 | 0.07 | 3.11 | 0.4329 |
| | | DC | 74 % | 84 % | 1.90 | 81 % | 79 % | 0.90 | 0.15 | 5.53 | 0.9074 |

TABLE 12
Descriptive Statistics of Structure and Size Attributes for the CC and DC Designs

| Measure | Description | Design | Median | Mean | Sum |
|---|---|---|---|---|---|
| *OMMIC_N(c)* | The number of method invocations from a (client) class *c* to non-library classes | CC | 2 | 4.7 | 33 |
| | | DC | 1 | 2.8 | 34 |
| *OMMIC_L(c)* | The number of method invocations from a (client) class *c* to library classes | CC | 0 | 1.1 | 8 |
| | | DC | 0 | 1.3 | 16 |
| *OMMEC(c)* | The number of method invocations to a (server) class *c* | CC | 3 | 4.7 | 33 |
| | | DC | 2 | 2.8 | 34 |
| *MC(c)* | The number of implemented methods in a class *c* | CC | 2 | 2.1 | 15 |
| | | DC | 2 | 2.4 | 29 |
| *CS(c)* | The size (in SLOC) of each class. Note that the sum corresponds to system size. | CC | 24 | 28.3 | 198 |
| | | DC | 22 | 23.9 | 287 |

explanation is that the DC design has many more methods, each of which do less work and delegate more, resulting in "delocalized plans" [37] that are difficult to comprehend for less experienced developers.

## ACKNOWLEDGMENTS

The international pair programming experiment was financed partly by the Research Council of Norway through the research projects INTER-PROFIT (INTERnational PROcess improvement For the IT industry) and SPIKE (Software Process Improvement based on Knowledge and Experience), and partly by Simula Research Laboratory. In particular, the support by the SPIKE project manager, Tor Ulsund, is very much appreciated. Furthermore, the authors thank KompetanseWeb and Gunnar Carelius for their support on the SESE tool, Hans Christian Benestad for excellent project administration and the recruitment of subjects for the experiment, Siw Elisabeth Hove for providing valuable support during the preparation and quality assurance of the experimental materials in SESE, Chris Wright for proofreading, and Are Magnus Bruaset and Sindre Mehus for their excellent work on the assessment of the Java solutions delivered by the subjects. The authors also thank the students at the University of Oslo who participated in three pilot experiments. Finally, this work would not have been possible without the consultants and project managers who participated from the following companies: Accenture, Avega AB, Bekk Consulting, Bluefish AB, Cap Gemini Norge, Cap Gemini Sverige AB, CIBER UK, Coredump AB, Dotify AB, Ementa, Ementor, ErgoGroup, Genera, Kantega, LostWax, Objectnet, Oracle Corporation UK, Qbranch Consulting AB, Software Innovation, Software Innovation Technology Sweden, Steria, Sun Microsystems Sweden, Synaptic AB, Systek, Systemfabrikken, TietoEnator, Unified Consulting, UPCO, and WM-data AB.

## REFERENCES

[1] E. Arisholm and D.I.K. Sjøberg, "A Controlled Experiment with Professionals to Evaluate the Effect of a Delegated versus Centralized Control Style on the Maintainability of Object-Oriented Software," Technical Report 2003-6, Simula Research Laboratory, http://www.simula.no/~erika, 10 Sept. 2003.

[2] E. Arisholm and D.I.K. Sjøberg, "Evaluating the Effect of a Delegated versus Centralized Control Style on the Maintainability of Object-Oriented Software," *IEEE Trans. Software Eng.*, vol. 30, no. 8, pp. 521-534, Aug. 2004.

[3] E. Arisholm, D.I.K. Sjøberg, G.J. Carelius, and Y. Lindsjørn, "A Web-Based Support Environment for Software Engineering Experiments," *Nordic J. Computing*, vol. 9, no. 4, pp. 231-247, 2002.

[4] K. Beck, "Embrace Change with Extreme Programming," *Computer*, vol. 32, no. 10, pp. 70-77, Oct. 1999.

[5] K. Beck and C. Andres, *Extreme Programming Explained: Embrace Change,* second ed. Addison Wesley, 2004.
[6] C.F. Bond and L.J. Titus, "Social Facilitation: A Meta-Analysis of 241 Studies," *Psychological Bull.,* vol. 94, no. 2, pp. 265-292, 1983.
[7] R. Brown, *Group Processes,* second ed. Blackwell, 2000.
[8] J.-M. Burkhardt, F. Detienne, and S. Wiedenbeck, "Object-Oriented Program Comprehension: Effect of Expertise, Task and Phase," *Empirical Software Eng.,* vol. 7, no. 2, pp. 115-156, 2002.
[9] L. Cao and P. Xu, "Activity Patterns of Pair Programming," *Proc. 38th Hawaii Int'l Conf. System Sciences,* 2005.
[10] A. Cockburn and L. Williams, "The Costs and Benefits of Pair Programming," *Extreme Programming Examined,* G. Succi and M. Marchesi, eds., Addison Wesley, 2001.
[11] J. Cohen, *Statistical Power Analysis for the Behavioral Sciences,* second ed. Lawrence Erlbaum Assoc., 1988.
[12] J. Cohen, "A Power Primer," *Psychological Bull.,* vol. 112, no. 1, pp. 155-159, 1992.
[13] L.L. Constantine, *Constantine on Peopleware.* Prentice Hall, 1995.
[14] T.D. Cook and D.T. Campbell, *Quasi-Experimentation—Design & Analysis Issues for Field Settings.* Houghton Mifflin, 1979.
[15] J.O. Coplien, "A Generative Development-Process Pattern Language," *Pattern Languages of Program Design,* J.O. Coplien and D.C. Schmidt, eds., pp. 183-237, Addison-Wesley, 1995.
[16] T. Dybå, V.B. Kampenes, and D.I.K. Sjøberg, "A Systematic Review of Statistical Power in Software Engineering Experiments," *Information and Software Technology,* vol. 48, no. 8, pp. 745-755, 2006.
[17] N.V. Flor and E.L. Hutchins, "Analyzing Distributed Cognition in Software Teams: A Case Study of Team Programming During Perfective Software Maintenance," *Proc. Fourth Workshop Empirical Studies of Programmers,* pp. 36-64, 1991.
[18] D.R. Forsyth, *Group Dynamics,* third ed., Wadsworth, 1999.
[19] H. Gallis, E. Arisholm, and T. Dybå, "An Initial Framework for Research on Pair Programming," *Proc. 2003 ACM-IEEE Int'l Symp. Empirical Software Eng. (ISESE '03),* pp. 132-142, 2003.
[20] S. Heiberg, U. Puus, P. Salumaa, and A. Seeba, "Pair-Programming Effect on Developers Productivity," *Proc. Int'l Conf. Extreme Programming and Agile Processes in Software Eng.,* pp. 215-224, 2003.
[21] S. Holm, "A Simple Sequentially Rejective Multiple Test Procedure," *Scandinavian J. Statistics,* vol. 6, pp. 65-70, 1979.
[22] T. Hærem, "Task Complexity and Expertise as Determinants of Task Perceptions and Performance," PhD dissertation, Norwegian School of Management BI, 2002.
[23] S.J. Karau and K.D. Williams, "Social Loafing: A Meta-Analytic Review and Theoretical Integration," *J. Personality and Social Psychology,* vol. 65, no. 4, pp. 681-706, 1993.
[24] B.A. Kitchenham, S.L. Pfleeger, L.M. Pickard, P.W. Jones, D.C. Hoaglin K.E. Emam, and J. Rosenberg, "Preliminary Guidelines for Empirical Research in Software Engineering," *IEEE Trans. Software Eng.,* vol. 28, no. 8, pp. 721-734, Aug. 2002.
[25] K.M. Lui and K.C.C. Chan, "When Does a Pair Outperform Two Individuals?" *Proc. Int'l Conf. Extreme Programming and Agile Processes in Software Eng.,* pp. 225-233, 2003.
[26] K.M. Lui and K.C.C. Chan, "A Cognitive Model for Solo Programming and Pair Programming," *Proc. Int'l Conf. Cognitive Informatics (ICCI '04),* 2004.
[27] C. McDowell, L. Werner, H. Bullock, and J. Fernald, "The Effects of Pair-Programming on Performance in an Introductory Programming Course," *Proc. 33rd SIGCSE Technical Symp. Computer Science Education,* pp. 38-42, 2002.
[28] R.H. Myers, D.C. Montgomery, and G.G. Vining, *Generalized Linear Models: With Applications in Engineering and the Sciences,* first ed. Wiley-Interscience, 2001.
[29] M.M. Müller, "Are Reviews an Alternative to Pair Programming?" *Empirical Software Eng.,* vol. 9, no. 4, pp. 335-351, 2004.
[30] 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.
[31] J. Nawrocki and A. Wojciechowski, "Experimental Evaluation of Pair Programming," *Proc. European Software Control and Metrics Conf. (ESCOM),* 2001.
[32] J.T. Nosek, "The Case for Collaborative Programming," *Comm. ACM,* vol. 41, no. 3, pp. 105-108, 1998.
[33] F. Padberg and M.M. Müller, "Analyzing the Cost and Benefit of Pair Programming," *Proc. Ninth Int'l Software Metrics Symp.,* pp. 166-177, 2003.
[34] W.R. Shadish, T.D. Cook, and D.T. Campbell, *Experimental and Quasi-Experimental Designs for Generalized Causal Inference.* Houghton-Mifflin, 2002.
[35] S.D. Sheetz, "Identifying the Difficulties of Object-Oriented Development," *J. Systems and Software,* vol. 64, no. 1, pp. 23-36, 2002.
[36] D.I.K. Sjøberg, B. Anda, E. Arisholm, T. Dybå, M. Jørgensen, A. Karahasanovic, and M. Vokác, "Challenges and Recommendations when Increasing the Realism of Controlled Software Engineering Experiments," *Empirical Methods and Studies in Software Engineering: Experiences from ESERNET,* R. Conradi and A.I. Wang, eds., pp. 24-38, Springer-Verlag, 2003.
[37] E. Soloway, R. Lampert, S. Letowski, D. Littman, and J. Pinto, "Designing Documentation to Compensate for Delocalized Plans," *Comm. ACM,* vol. 31, no. 11, pp. 1259-1267, 1988.
[38] B.W. Tuckman, "Development Sequence in Small Groups," *Psychological Bull.,* vol. 63, no. 6, pp. 384-399, 1965.
[39] M. Vokác, W. Tichy, D.I.K. Sjøberg, E. Arisholm, and M. Aldrin, "A Controlled Experiment Comparing the Maintainability of Programs Designed with and without Design Patterns—A Replication in a Real Programming Environment," *Empirical Software Eng.,* vol. 9, no. 3, 2004.
[40] G.M. Weinberg, *The Psychology of Computer Programming.* Van Nostrand Reinhold, 1971.
[41] L. Williams and R.L. Upchurch, "In Support of Student Pair-Programming," *Proc. 32nd SIGCSE Technical Symp. Computer Science Education,* pp. 327-331, 2001.
[42] L. Williams and R. Kessler, *Pair Programming Illuminated.* Addison-Wesley, 2002.
[43] L. Williams, A. Shukla, and A.I. Antón, "An Initial Exploration of the Relationship between Pair Programming and Brooks' Law," *Proc. Agile Development Conf. (ADC '04),* pp. 11-20, 2004.
[44] L. Williams, R.R. Kessler, W. Cunningham, and R. Jeffries, "Strengthening the Case for Pair Programming," *IEEE Software,* vol. 17, no. 4, pp. 19-25, 2000.
[45] L. Williams, E. Wiebe, K. Yang, M. Ferzli, and C. Miller, "In Support of Pair Programming in the Introductory Computer Science Course," *Computer Science Education,* vol. 12, no. 3, pp. 197-212, 2002.
[46] L.A. Williams, "The Collaborative Software Process," PhD dissertation, Univ. of Utah, 2000.
[47] R.J. Wirfs-Brock, "Characterizing Your Application's Control Style," *Report on Object Analysis and Design,* vol. 1, no. 3, 1994.
[48] R.J. Wirfs-Brock and B. Wilkerson, "Object-Oriented Design: A Responsibility Driven Approach," *SIGPLAN Notices,* vol. 24, no. 10, pp. 71-75, 1989.
[49] C.F.J. Wu and M. Hamada, *Experiments: Planning, Analysis, and Parameter Design Optimization,* first ed. Wiley-Interscience, 2000.
[50] R.B. Zajonc, "Social Facilitation," *Science,* no. 149, pp. 269-274, 1965.

**Erik Arisholm** received the MSc degree in electrical engineering from University of Toronto and the PhD degree in computer science from the University of Oslo. He has seven years of industry experience in Canada and Norway as a lead engineer and design manager. He is now the research project manager of the OOAD project in the Department of Software Engineering, Simula Research Laboratory, and an associate professor in the Department of Informatics, University of Oslo. His main research interests are empirical studies of object-oriented analysis and design principles, design quality measurement and prediction, and software process improvement. He is a member of the IEEE and the IEEE Computer Society.

**Hans Gallis** received the MSc degree in computer science from the University of Oslo in 2002 and then started the PhD degree in software engineering at the University of Oslo and Simula Research Laboratory. He worked as an IT consultant for two years before he started on his MSc studies. He is now on leave from Simula Research Laboratory to establish the company Symphonical AS. His research interests include empirical software engineering and software process improvement with a focus on agile software methodologies in general and collaboration and communication in particular (e.g., pair programming).

**Tore Dybå** received the MSc degree in electrical engineering and computer science from the Norwegian Institute of Technology in 1986 and the PhD degree in computer and information science from the Norwegian University of Science and Technology in 2001. He is the chief scientist at SINTEF ICT and a visiting scientist at the Simula Research Laboratory. He worked as a consultant for eight years in Norway and Saudi Arabia before he joined SINTEF in 1994. His research interests include empirical and evidence-based software engineering, software process improvement, and organizational learning. He is the author or coauthor of more than 50 publications appearing in international journals, books, and conference proceedings in the fields of software engineering and knowledge management. He is the principal author of the book *Process Improvement in Practice: A Handbook for IT Companies*, published as part of the Kluwer International Series in Software Engineering. He is a member of the International Software Engineering Research Network, the IEEE Computer Society, and the editorial board of *Empirical Software Engineering*.

**Dag I.K. Sjøberg** received the MSc degree in computer science from the University of Oslo in 1987 and the PhD degree in computing science from the University of Glasgow in 1993. He has five years of industry experience as a consultant and group leader. He is now a research director in the Department of Software Engineering, Simula Research Laboratory, and a professor of software engineering in the Department of Informatics, University of Oslo. Among his research interests are research methods in empirical software engineering, software processes, software process improvement, software effort estimation, and object-oriented analysis and design. He is a member of the IEEE and the IEEE Computer Society.