# An Empirical Study about the Feelgood Factor in Pair Programming

Matthias M. Müller and Frank Padberg
Fakultät für Informatik
Universität Karlsruhe
Am Fasanengarten 5, 76128 Karlsruhe, Germany
{muellerm|padberg}@ipd.uka.de

## Abstract

*Why are programmer pairs more productive than single developers ? Using empirical data from two controlled experiments, we find that pair performance is uncorrelated with programming experience, but shows a significant correlation with how comfortably the developers feel with pair programming during the session (the "feelgood" factor).*

## 1. Introduction

Pair programming has been the subject of a number of empirical studies over the last two years. During pair programming, all tasks are performed by pairs of developers using one keyboard, display, and mouse. There is empirical evidence that pairs are considerably more productive than single developers; in addition, the code written by programmer pairs seems to be of higher quality (see the next section for references).

Why is a pair of programmers, considered as a "unit," more productive than a single developer ? Why are some pairs more productive than others ? Can we estimate the productivity of a pair by measuring certain features of the pair ? Categories of possible features include :

- features describing the performance of the individual developers who form the pair, such as their programming experience or productivity;
- features describing the interaction between the developers when pair programming, for example, how smoothly they communicate or how comfortably they feel during the pair sessions ("soft factors").

In this empirical study, we take a closer look at two orthogonal features of a pair which potentially could drive its productivity. We raise the following two questions :

**Q1** Is there empirical evidence that the performance of a pair is dominated by the programming experience of the developers ?

**Q2** Is there empirical evidence that the performance of a pair is dominated by how comfortably the developers feel during the pair session ?

We call the latter feature (how comfortably the developers feel in a pair session) the **feelgood factor** of pair programming. We'd like to point out that we view these questions as research hypotheses and our study as a first contribution to their empirical investigation.

The data for this study comes from two controlled experiments with 38 subjects [3, 4]. The pre-test questionnaire asked for various measures for the programming experience of the subjects (in years and lines of code). The post-test questionnaire asked for the feelgood factor, see SECTION 3. Despite being self-reported, these measures carry valuable information which is hard to measure in an objective way. We lack data about the productivity of the developers. We deliberately factored out the code quality aspect by guaranteeing a uniform, high quality level for each pair through external acceptance tests.

On our data-set, we get the following results.

- Pair performance is uncorrelated with the programming experience.
- The feelgood factor is a candidate driver for the performance of a pair.

These results are preliminary due to the small size of our data set.

For the first result, we studied different measures of the experience level of the two individuals in a pair, as well as the corresponding mean value, which we use as a measure for the experience of the pair. None of these measures correlated with the implementation time of the programming tasks.

The second result is based on a negative statistical correlation between the feelgood factor of a pair and the implementation time for the tasks. From the correlation alone, one can *not* decide whether a pair performed well because the feelgood factor was high, or, whether the developers felt comfortable with pair programming because they had the impression that they were performing well. To answer that question, further empirical studies are necessary. In our experiments, most subjects had no prior experience with pair programming. All subjects had a positive attitude towards the technique. This fact might be an indication that the pair performance depends on the feelgood factor, and not the other way around.

## 2. Empirical Results about the Benefit of Pair Programming

The difference in development speed between a conventional project and a pair programming project is measured by a special process metric, the *PairSpeedAdvantage* (PSA). The PSA is defined as the ratio between the time required for some task by an average single developer in a conventional project and the time required by a pair of programmers. The empirical studies available today indicate that the PSA ranges between $1.0$ and $1.7$ [1, 6, 7, 9]. For example, Nosek [7] reports that programmer pairs on average require a $29$ percent shorter time to completion for their tasks than single programmers. Using Nosek's data, the speed advantage equals

$$\mathsf{PSA} = \frac{100}{100 - 29} = 1.4.$$

By definition of the pair speed advantage, the productivity of a pair equals the (average) productivity of single developer multiplied by the pair speed advantage.

Although pairs are faster than single developers, they in general are not twice as fast. To analyze the tradeoff between the cost and benefit of pair programming, we have presented an economic model for pair programming (and Extreme Programming) in previous papers [8, 5]. The results of our economic studies show that pair programming makes sense *economically* only if the market pressure is high. In that case, entering the market faster covers the extra personnel cost.

Besides an increased productivity, there can be other good reasons for using pair programming. Potential benefits which are discussed in the literature include higher code quality [3, 8, 9] and training of developers [2, 10]. The training aspect is particularly interesting in educating undergraduate computer science students.

## 3. The Controlled Experiments

The data for this study were collected during two controlled experiments. The experiments were held during the summer terms 2002 and 2003, and are labeled Exp02 and Exp03. The original purpose of the experiments was to investigate whether the average effort of pairs to complete a programming assignment exceeds the average effort of single developers assisted by an additional review phase. That purpose led to a counterbalanced design of the experiments. In this paper, we use the experiment data (including the pre- and post-test questionnaires) to study questions Q1 and Q2 raised in the introduction.

### 3.1. Environment

Each experiment was part of an extreme programming course. The courses consisted of four short sessions, which introduced key Extreme Programming (XP) techniques (pair programming, test-first, refactoring, and the planning game), and a whole week of project work. The session introducing pair programming was taught by XP professionals from industry and took about 1.5 hours. The experiments took place between the introductory sessions and the project week. The XP techniques other than pair programming were not applied in the experiments.

A total of 38 subjects participated in the experiments, yielding 19 pairs. The subjects subscribed voluntarily to the XP course. Before signing up for the course, they knew that they had to take part in an experiment in order to get the course credits. All subjects were computer science students after the "Vordiplom" who were on average in their fourth year of study, see FIGURE 1.

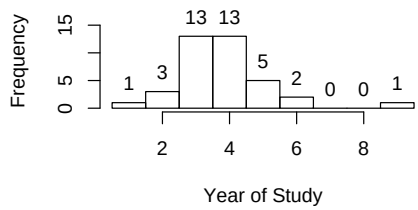


**Figure 1. Distribution of year of study of the subjects.**

Java was the programming language for both the experiment and the lab course.

### 3.2. Tasks

Due to the counterbalanced design of the experiments, the subjects solved two different tasks:

**Polynomial** Find the zero positions of an arbitrary polynomial of third degree. The subjects had to implement the method `findZeroPosition` of a given class `Polynomial`.

**Shuffle-Puzzle** Find the solution of a given shuffle-puzzle within a given number of moves and list the moves, if a solution exists. The subjects had to add a method `findMoves` to the basic class `ShufflePuzzle`.

The classes `Polynomial` and `ShufflePuzzle` already contained constructors and methods for I/O to facilitate implementation and final testing.

The description of the task `Polynomial` contained a hint for a possible numeric solution to the problem. However, the students were not forced to use a special method to solve the problem; they could use any method which they considered suitable for the problem. As a special difficulty, the task required careful handling of the floating point arithmetic. For most students, solving the task involved implementing the suggested method as well as taking care of special cases. The shuffle-puzzle task required solving a backtracking problem which the students knew how to solve from their first computer-sciences courses.

In the experiments, one task was solved alone with an additional review phase; for the other task, the subjects paired off. Therefore, half the pairs worked on `Polynomial`, the other half on `Shuffle-Puzzle`. The two programming tasks are of similar complexity: FIGURE 2 shows box plots for the time which the pairs needed for completion of the tasks (9 pairs for `Polynomial`, 10 pairs for `Shuffle-Puzzle`).

Except for an outlier in `Shuffle-Puzzle`, there is a large overlap between the `Polynomial` and the `Shuffle-Puzzle` completion time distributions. Hence, it is justified to treat the two tasks as practically equivalent in this study. This is also supported by a p-value of 0.87 in the corresponding two-sided Wilcoxon test.

### 3.3. Plan

The procedure for the pair programming task is outlined in FIGURE 3. The procedure is divided into a *coding and testing* and a *quality assurance* phase. During coding and testing, the pairs worked on the assignment until they felt they were done. Then, they entered quality assurance where their program had to pass 95 out of 100 test cases of an external acceptance test. If the program passed, the pair was done. If the program did not reach the required 95 percent

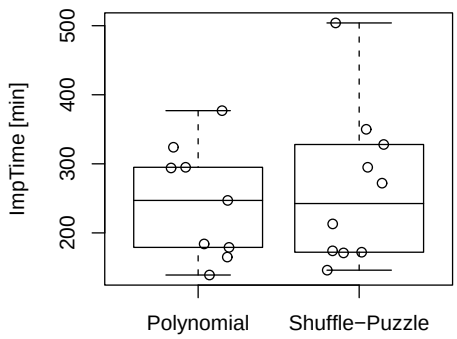


**Figure 2. Implementation times for the two tasks.**

correctness, the pair received the output of the failed tests and had to fix the errors. The acceptance test and the rework phase were repeated until the program passed 95 tests.

The aim of the quality assurance phase and the acceptance test was to guarantee a high and uniform code quality in the experiment. The individual attitude towards testing and program quality is factored out. However, the exit criterion of the quality assurance phase still leaves some room for variation in program reliability. A comprehensive additional acceptance test with 700,000 test cases for `Polynomial` and 15,000 test cases for `Shuffle-Puzzle` shows that for both tasks a high code quality was achieved: for `Polynomial`, the minimum correctness in the large test was 89 percent with a median of 93 percent; for `Shuffle-Puzzle` the minimum correctness was 99 percent.

The pair programming procedure could be done in one session. For each session, the pairs made an appointment with the experimenter. If the task could not be finished in the first run, a subsequent appointment had to be made. Short breaks were permitted, for example, for smoking or going to lunch. During the breaks, the clock was stopped.

### 3.4. Selection of Pairs

The two controlled experiments had counterbalanced designs since we originally wanted to compare pair programming against reviews. This particular design does not really matter for the present study, though. In the first experiment (Exp02), 2 out of 20 subjects had prior experience with pair programming (one and nine months). In the second experiment (Exp03), 6 out of 18 subjects had pair programming experience between one and six months. The first experi-

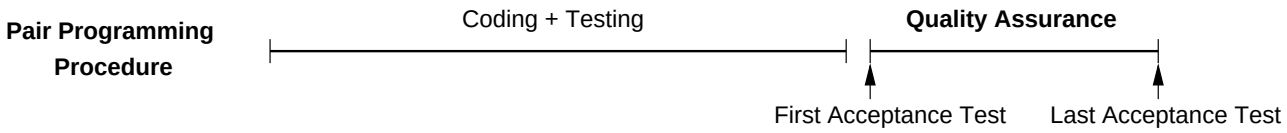


**Figure 3. Procedure for pair programming task.**

ment and its results are described in [3]; the results of the combined experiments are presented in [4].

We have a total of 19 data points. TABLE 1 shows the number of pair programming data points for each experiment and task.

**Table 1. Number of data-points in each experiment (Pol=Polynomial, Shu=Shuffle-Puzzle).**

| | Pol | Shu | total |
|---|---|---|---|
| Exp02 | 5 | 5 | 10 |
| Exp03 | 4 | 5 | 9 |
| total | 9 | 10 | 19 |

The pairing of the subjects in the experiments was done according to their overall programming experience in lines of code, independent of the programming language. The subjects had reported their programming experience in the pre-test questionnaire, see the next section. In each experiment, the most skilled subject had to pair off with the lowest skilled subject, the second best skilled subject with the second lowest skilled subject, and so on. The aim was to balance the skill level across the pairs somewhat.

## 3.5. Measured Data

The data used in this study originate from three different sources: the pre-test questionnaire; data measured during development; and the post-test questionnaire.

### 3.5.1. Pre-Test Questionnaire

Before the subjects paired off, they had to fill out the pre-test questionnaire. The pre-test questionnaire asked for various measures of the individual programming experience, see FIGURE 4.

The data obtained from the questionnaire is subjective. The programming experience is our only means in this study for assessing individual programming skills. Individual productivity figures of the subjects would have been a better means for quantifying their programming skills. The productivity could have been measured with an additional test for each student before the experiment took place. Since the experiment already meant a considerable overhead for

Pre-Test Questionnaire

How much programming experience do you have (in years) ? _____

How many lines of code did you develop ?
◯ less than 3,000
◯ less than 10,000
◯ less than 40,000
◯ more than 40,000

How much Java programming experience do you have (in years) ? _____

How much Java programming experience do you have (in lines of code) ? _____

**Figure 4. Relevant questions in pre-test questionnaire.**

the students, we abstained from such a test to avoid further overhead. Due to the lack of productivity data, all we can use in this study are the subjective programming experience data collected in the pre-test questionnaire.

Based on the pre-test data, we compute the following measures for the experience of a pair:

- mean overall programming experience of a pair in years (PairProgExp);
- mean Java programming experience of a pair in years (PairJavaExp);
- mean Java programming experience of a pair in lines of code (PairJavaLOC);

As can be seen from FIGURE 5, the experience level of the pairs has a large range for each of these metrics, despite our special scheme for pairing the subjects (see SECTION 3.4).

In the box plot for the Java experience of a pair measured in lines of code, an outlier with a value of 127,000 lines of code is left out. The outlier corresponds to a pair with self-reported 250,000 and 4,000 lines of Java code. In addition

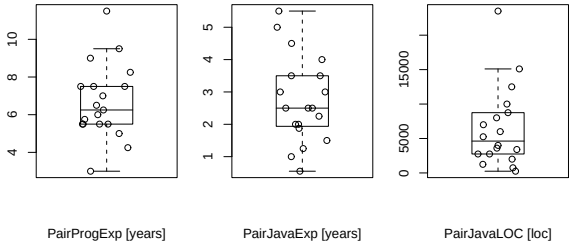


**Figure 5. Distribution of pair experience.**

to being a graduate student, the experienced subject in this pair is a developer in his own small software firm.

### 3.5.2. Implementation Time

For the pair programming task, two different periods of time were measured, see FIGURE 3: the time spent for coding and testing ($T_{CT}$) and the time spent for quality assurance ($T_{QA}$). The time for quality assurance consists of all the rework time until the acceptance test was passed; the execution time for the acceptance tests is not included. Thus, the implementation time (ImpTime) for a task equals:

$$\mathsf{ImpTime} = T_{CT} + T_{QA}.$$

The implementation time reflects the elapsed time for a programmer pair to finish the assignment at the prescribed quality level. For the purpose of this study, we use the implementation time as a measure for the performance of a pair.

FIGURE 6 shows the distribution of the implementation time for the combined experiments.

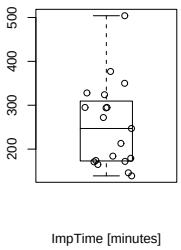


**Figure 6. Distribution of implementation time.**

### 3.5.3. Post-Test Questionnaire

After the pair programming experiment, each student was handed out the post-test questionnaire, see FIGURE 7. The post-test questionnaire asked for how comfortable the student felt during the pair programming session. The answer ranges on an ordinal scale.

Post-Test Questionnaire

How did you like pair programming ?
(1=not at all, 5=very much)

1:◯ 2:◯ 3:◯ 4:◯ 5:◯

**Figure 7. Relevant question in post-test questionnaire.**

We call this metric the *individual* feelgood factor of a developer. Again, the data obtained from the questionnaire is subjective.

Since our questionnaire did *not* ask the pairs to specify a *joint* feelgood factor, we took the mean of the individual assessments as a substitute. We call the resulting metric the *pair feelgood factor*. The pair feelgood factor ranges between 2.5 and 5.0. The following box plot (FIGURE 8) shows the distribution of the pair feelgood factor.

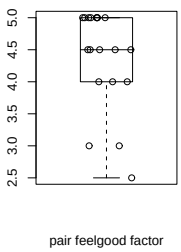


**Figure 8. Distribution of pair feelgood factor.**

Since the feelgood factor is defined as the mean of the individual assessments, FIGURE 8 shows that in most cases there was a close agreement in the assessments given by the individuals.

In future studies, it is desirable to measure the feelgood factor more frequently during the experiment than we did in this study because of the relatively short length of the pair programming sessions in our experiments.

## 3.6. Threats to Validity

Several perils threat the validity of our study. First, our data base is fairly small. Hence, it is difficult to draw decisive conclusions. Second, the subjects' individual attitude may have been biased in favor of pair programming. The answers to the question in the post-test questionnaire (see FIGURE 7) are shifted towards the right hand side of the scale, see FIGURE 9.

Third, subjects participated voluntarily in the course, so they might have felt more enthusiastic about pair pro-

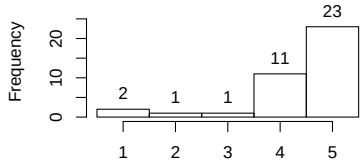


**Figure 9. Histogram for answers on how much individuals liked pair programming (1=not at all, 5=very much).**

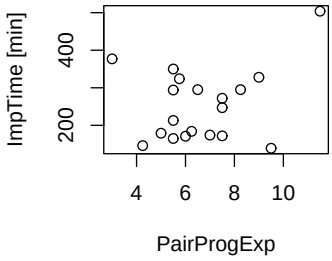


**Figure 10. Implementation time and pair programming experience in years.**

gramming than the average developer would have. Fourth, our particular selection strategy (see Section 3.4) resulted in some unbalanced pairs and some more evenly matched pairs; this might influence the feelgood factor if programmers prefers to be matched with a partner of similar experience. Fifth, the experiments took place in our lab. Thus, the students could have been aware of the progress of the other teams during the experiment. Although the pairs started to work on their assignments at any time, this threat can not be ignored. And finally, most subjects had no previous experience with pair programming. Subjects were assigned to a pair only according to their programming experience, ignoring any existing acquaintanceship with one another which might have improved their pair programming performance.

## 4. Results

In the analysis, we first study the correlation between the implementation time and the programming experience of the pair, respectively, the feelgood factor of the pair. In a second step, instead of looking at the pairs, we focus on the individual programming experience and the individual feelgood factor.

### 4.1. Experience of the Pair

To study the impact of the programming experience on the performance of the pair, we use the metrics PairProgExp, PairJavaExp, and PairJavaLOC, described in SECTION 3.5.1, as independent variables. The implementation time, as computed in SECTION 3.5.2, is the dependent variable.

The following scatter plot (FIGURE 10) shows the relationship between the experience level of the pairs and the implementation time needed for the tasks.

Apparently, there is no correlation between the experience level and the implementation time. This observation is also supported by a p-value of 0.68 in the Spearman test. Thus, the hypothesis that the variables PairProgExp and ImpTime are uncorrelated *cannot* be rejected at any reasonable significance level.

We get the same result when using the Java programming experience (measured either in years or lines of code) instead of the overall programming experience, see the scatter plots in FIGURE 11.

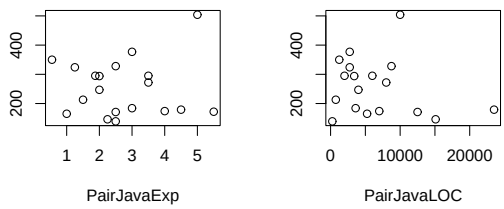


**Figure 11. Implementation time (in minutes) and pair Java programming experience (in years and lines of code). In PairJavaLOC an outlier at (127 000; 172) is omitted**

The corresponding p-values in the Spearman test are 0.85 and 0.33.[1]

For our data set, these results suggest that a pair's programming experience and pair performance are uncorrelated. This gives a negative answer to question Q1 from the introduction.

### 4.2. Feelgood Factor of the Pair

The scatter plot in FIGURE 12 shows the relation between the pair feelgood factor (PairFeelgood) and the implementation time (ImpTime).

[1]The p-value of 0.33 for the Java experience measured in lines of code does include the outlier at (127 000; 172).

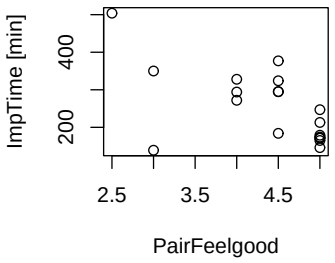


**Figure 12. Implementation time and pair feelgood factor.**

The scatter plot indicates that the pair performance and the pair feelgood factor might be correlated. Despite a few outlying data points, this observation is supported by a p-value of 0.01 in the corresponding Spearman test.

Thus, the hypothesis that the variables PairFeelgood and ImpTime are *un*-correlated can be rejected, for example, at the 5 percent level.

The fact that a correlation exists is *not* sufficient to conclude that the feelgood factor actually *drives* the pair performance: it is unclear whether a pair performs well because the feelgood factor is high, or, whether the developers feel comfortable because they have the impression that they are performing well. In particular, we are not yet in the position to answer question Q2 raised in the introduction. A valid conclusion is, though, that the pair feelgood factor is a *candidate* driver for the performance of a pair.

## 4.3. Individual Experience

We have seen in subsection 4.1 that the experience level of a pair does not correlate with the implementation time. It is natural to ask whether the same holds for the experience of the individual developers in a pair. To that end, we focused on the relationship between

- the implementation time and that member of each pair who had the *higher* overall experience level in years (HigherProgExp);
- the implementation time and that member of each pair who had the *lower* overall experience level in years (LowerProgExp).

The scatter plots in FIGURE 13 indicate that there is no correlation between these individual experience levels and the implementation time. The plots look similar when using the Java experience (in years or lines of code) instead of the overall programming experience.

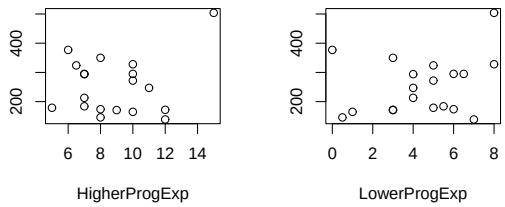


**Figure 13. Implementation time (in minutes) and higher experience level (in years).**

Our finding that the individual experience levels and the pair performance are uncorrelated is also supported by the p-values of the corresponding Spearman test, see TABLE 2.

**Table 2. P-values for correlation between individual programming experience and implementation time.**

| | higher | lower |
|---|---|---|
| overall experience [years] | 0.48 | 0.31 |
| Java experience [loc] | 0.25 | 0.40 |
| Java experience [years] | 0.56 | 0.74 |

## 4.4. Individual Feelgood Factor

We have seen in subsection 4.2 that the performance of a pair correlates with the feelgood factor of the pair. Thus, it is worthwhile to study whether this correlation is driven by one of the individuals in the pair.

We analyzed the relationship between the implementation time and that member of each pair who felt *less* comfortable with the pair programming situation than the other member (LowerFeelgood), see the left diagram in FIGURE 14.

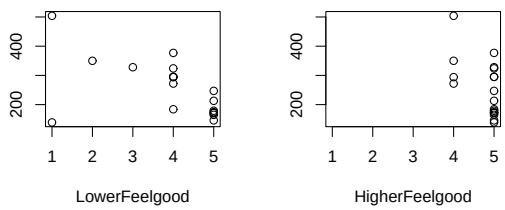


**Figure 14. Implementation time (in minutes) and individual feelgood factor.**

The scatter plot and the Spearman test indicate that the performance of the pair *might* be driven by the team member who feels less comfortable (p-value of 0.01). Again, the correlation alone is not sufficient to decide which variable depends on the other. Therefore, the existence of a correlation should be considered as a basis for future research.

We also analyzed the relationship between the implementation time and that member of each pair who felt *more* comfortable with the pair programming situation than the other member (HigherFeelgood), see the right diagram in FIGURE 14. The Spearman test indicates that there may or may not be a correlation, depending on the choice of the significance level (p-value equals 0.06). However, the feelgood factor takes on the values 4 and 5 only. Since only the upper range of the possible values is covered, one might question whether a correlation analysis really makes sense here.

## 5. Conclusions

In this paper, we have analyzed empirical data from two controlled pair programming experiments. We have studied to which degree the performance of a pair is correlated with the programming experience of the pair and the pair feelgood factor. We found that pair performance is uncorrelated with a pair's programming experience. We also found that the pair feelgood factor is a *candidate* driver for the pair performance. A similar hypothesis holds for the feelgood factor of that pair member who feels less comfortable with pair programming.

Some of our results may seem rather obvious; nonetheless, in the context of pair programming there still is a severe lack of empirical data. We view our study as a contribution in this direction.

Our findings are preliminary due to the small size of our data set. Besides measuring other features of the pair programming process, future studies should seek to include subjects who have more mixed feelings about pair programming than our students. Subjects also should have some prior experience with pair programming when participating in the experiment. The goal is to get a more balanced data set. In addition, we need metrics which more directly measure features of a programmer pair, such as a direct measure for the current condition of a pair. One would also like to ask several questions from different perspectives about the feelgood factor in order to get a more differentiated picture.

Based on our empirical results, we would suggest the following guideline: *First of all, make your pairs feel good !* It seems to be important that both developers in a pair feel comfortable with the pair programming situation and remain closely in sync. We'd like to point out that our data only yields a correlation between the pair performance and its feelgood factor; therefore, we currently cannot decide whether a pair performs well because the feelgood factor is high, or, whether the developers feel comfortable because they have the impression they are performing well.

We consider it an open question how much the performance of a pair suffers from a large gap between the two individuals in their productivity, experience, or social skills. In particular, it might be the case that pair performance is driven by the feelgood factor of the developer who feels less comfortable with pair programming.

Further research in this area potentially leads to ...

- a better understanding of the (social) mechanisms underlying pair programming;
- management guidelines how to best select the members of a pair and train developers in pair programming;
- early indicators for performance problems in a pair.

## References


[1] A. Cockburn and L. Williams. The costs and benefi ts of pair programming. In *eXtreme Programming and Flexible Processes in Software Engineering (XP2000)*, Cagliari, Italy, June 2000.

[2] C. McDowell, L. Werner, H. Bullock, and J. Fernald. The effects of pair-programming on performance in an introductory programming course. In *SIGCSE Technical Symposium on Computer Science Education*, pages 38–42, Cincinnati, Kentucky, USA, 2002.

[3] M. Müller. Are reviews an alternative to pair programming ? In *Empirical Assessment In Software Engineering (EASE)*, pages 3–14, Keele, UK, Apr. 2003.

[4] M. Müller. Should we use programmer pairs or single developers for the next project ? Technical Report 2004-8, Faculty of Informatics, Universität Karlsruhe, 2004.

[5] M. Müller and F. Padberg. On the economic evaluation of xp projects. In *Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, pages 168–177, Helsinki, Finland, Sept. 2003.

[6] J. Nawrocki and A. Wojciechowski. Experimental evaluation of pair programming. In *European Software Control and Metrics (Escom)*, London, UK, 2001.

[7] J. Nosek. The case for collaborative programming. *Communications of the ACM*, 41(3):105–108, Mar. 1998.

[8] F. Padberg and M. Müller. Analyzing the cost and benefi t of pair programming. In *International Symposium on Software Metrics (Metrics)*, Sydney, Australia, Sept. 2003.

[9] L. Williams, R. Kessler, W. Cunningham, and R. Jeffries. Strengthening the case for pair-programming. *IEEE Software*, pages 19–25, July/Aug. 2000.

[10] L. Williams, C. McDowell, N. Nagappan, J. Fernald, and L. Werner. Building pair programming knowledge through a family of experiments. In *International Symposium Empirical Software Engineering (ISESE)*, pages 143–152, Rome, Italy, Sept. 2003.