Perceived Effects of Pair Programming in an Industrial Context

Jari Vanhanen and Casper Lassenius
Helsinki University of Technology, Software Business and Engineering Institute
P.O.BOX 9210, 02015 TKK, Finland
{firstname.lastname}@tkk.fi

Abstract

We studied the perceived effects of pair programming (PP) compared to solo programming in a large scale, industrial software development context. We surveyed developers (N=28) regarding effects of PP on learning, quality, effort, schedule, and human factors. Our findings support earlier results from studies done with students, or professionals doing small tasks. The positive effects of PP were largest for learning, schedule adherence of tasks, getting to know other developers, and team spirit. A small but clearly positive effect was perceived for various quality aspects, discipline in following work practices, and enjoyment of work. The improvement of estimation accuracy was almost negligible. The amount of refactoring did not change. On the negative side, the development effort for individual features was higher. In the beginning of the adoption, the exhaustiveness of work was perceived higher, but over time it decreased to the level of solo programming.

1. Introduction

In *pair programming* (PP) two persons design, code and test software together at one computer. The *driver* controls the keyboard. The *navigator* observes the driver's work trying to find defects, but also thinks at a more strategic level. The persons communicate actively and switch roles periodically. [1]

A global survey of the adoption of various development practices reported that 35% of development projects used PP [2]. The interest in PP is natural because PP can have a positive effect on, e.g., quality of code and design, knowledge transfer, learning, team work, and work satisfaction [3,4,5]. The negative effects can be increases in development effort [3,4,6,7] and mental exhaustiveness of work [1,8].

The existence and magnitude of the effects of PP may be influenced by many context factors such as characteristics of the developers, roles, communication, partner switching, type of work, development process and tools, and workspace facilities [12]. The effects of PP have been studied mostly with students who have made small, isolated tasks [6] or small projects [4,5,9]. In a few studies the subjects have been professionals, but in these, PP has been used only for small tasks [3,10,11]. Thus, the context has differed from realistic situations, where, e.g., a large, co-located team of senior developers develops a large system in a long project as a full-time job. Experiences of applying PP in the industry have been reported [8,13,14,15,16,17,18] but only two papers [13,18] report measured data about the effects of PP compared to solo programming. Thus, while previous research sheds some light on the effects of PP, very little has been reported regarding them in the industry.

We collected developers' perceptions of the effects of PP compared to solo programming in a company during a two year period when they adopted PP. We also report the developers' experiences of some practicalities of PP. The experiences are valuable for anyone interested in adopting PP, but also important for the study due to the context specificity of the effects of PP.

Chapter 2 summarizes the previous PP literature. Chapter 3 introduces the research methodology and the case organization. Chapter 4 characterizes the use of PP in the case organization. Chapter 5 analyzes the effects of PP, and Chapter 6 concludes the paper.

2. Related work

This chapter summarizes the proposed effects of PP. The goal is to list all potential effects having relevance in the industry in order to create a comprehensive reference list. Therefore, we do not evaluate the reliability of the existing evidence or the theories behind the proposed benefits. In the second section we discuss industrial experiences of the practicalities of PP, because the details about pair formation, partner combinations and role switching may affect the effects of PP.

2.1. Effects of pair programming

Here we summarize the proposed effects of PP classified under learning, quality, effort and schedule, and human factors (Table 1). The studies are still inconclusive, and the summary should be taken as a long list of potential effects rather than a list of proven effects.

Williams and Kessler [1] propose that PP improves communication especially when pairing with several different people. Learning about the developed system and development tools increases as the partners exchange knowledge of them. Increases in knowledge transfer have been found in one study [5].

Constantine noticed in a company that pairs produced "*nearly 100% bug free code, which was also better, tighter and more effective*" [19]. Beck proposes that PP produces higher quality code, and the pairs are more disciplined in following agreed work practices [20]. Williams and Kessler say that the navigator finds many defects early, and the pairs end up with better solutions to problems. Van Deursen suggests that PP benefits program comprehension since it produces better code and developers learn the code and program understanding strategies from their partners [21]. Improvements in quality metrics, e.g. test case pass rate, have been found in several studies [3,4,10,13,18,22] but in one study [6] no effect was found.

Nosek found that pairs used shorter elapsed time than individuals, but 42% more effort [3]. He proposes that PP could be utilized to speed up development. Williams and Kessler [1] propose that the total effort between pairs and individuals is equal. In various studies the effects on effort have varied between −28% and +100% [3,4,5,6,7,9,10]. Large differences in the contexts and experimental settings probably explain the different results. For example, in a study [7] the effort increase varied from 29% in difficult work to 91% in easy work. In another study [10] the results were opposite; the effort increase was 112% in complex work and 60% in easy work. The smallest effort increases [5,9] and even effort decreases [4,13,18] have been reported from team/project contexts. PP has been found to be more predictable regarding the effort spent [6] and to improve schedule adherence [13].

PP may affect several human factors. Pairs have more courage to do difficult things such as refactoring complex code [1,20]. PP gets people to know each other and thus builds trust and improves teamwork [1]. Programmers like PP, a fact that could improve morale and decrease personnel turnover [1]. Improvements in developers' confidence in their solutions and enjoyment of work have been confirmed in some studies [3,4,5,23]. On the other hand, PP can be mentally exhaustive [1,8].

Table 1. Proposed effects of pair programming

Area	Effect
Learning	- increased learning of work related topics
Quality	- smaller number of defects - better solutions to problems - higher comprehensibility of code and design
Effort and schedule	- increased total effort for a task - more accurate effort estimates - finishing tasks faster - better schedule adherence
Human factors	- higher satisfaction and confidence in work - higher exhaustiveness of work - improved trust and teamwork - more discipline - more courage to attack difficult things, such as refactoring, and admitting ignorance

2.2. Practicalities of pair programming

The adoption of PP may start slowly due to the expected increase in effort [14,15]. Successful ways for increasing the use of PP have included explicitly reserving time for PP [15], persuasion by management [14], keeping two developers responsible for the quality of a task [14], rewarding use of PP in the form of avoiding formal reviews [13], and dropping the ownership of work stations [11]. The amount of PP use has seldom been reported. Figures of 30% [8] and 50% [16] for time spent with a pair have been reported in XP projects, where PP is supposed to be used for all development work. In an experiment [4] students were supposed to use PP for all work, but in practice they used it for about 90% of the planning, design and testing activities and about 75% of the coding activity. In general, PP seems to be most applicable to more complex work such as design and complex coding tasks [11,14,15,16].

The pairs are often formed casually, e.g. in daily meetings [24,25,26], when new tasks arrive [13], or when an appropriate partner combination is discovered through overhearing what others are doing [24,27]. Pairs have also been rotated after a fixed time, e.g. one day, or even every couple of hours [17].

Some issues with partner combinations have been reported, e.g., a person having excess ego [1], or large differences in age [16]. However, the experiences are quite limited and to some degree contradictory. For example, issues have been reported both when partners have similar [18,28] and different expertise [16,28].

Switching driver/navigator roles periodically is important, because it activates a possibly passive navigator by letting him write [1]. In practice, frequent role switching does not always seem to realize [25].

3. Research method

This chapter introduces the case background and describes the research goals and questions along with the data collection and analysis methods.

3.1. Research goal and questions

Our research goal was to increase understanding of the effects of PP in a realistic, industrial context. Because the effects are context specific [12], we additionally studied in detail how the company applied PP. The research question was: *What are the perceived effects of PP compared to solo programming?* We scoped the question to cover 18 aspects of learning, quality, effort and schedule, and human factors (Figures 1-4) derived from the proposed effects of PP (Table 1).

3.2. Case description

The case organization was a department in a medium-sized Finnish software product company. At the end of the study, the department had a few dozens of developers divided into several development teams, each having a senior developer as a team leader. All developers were sitting in the same open office containing many cubicles and desks. The department was responsible for the development of a large and very successful software product, with a development history of 15 years. The development languages were C/C++, and the most widely used development environment was Visual Studio.

The motivation for adopting PP was to improve software quality and increase knowledge transfer among developers. The department had a PP champion that the first author helped in a small way in introducing PP to the developers, and in creating PP guidelines for them. The guidelines suggested, e.g., using PP in particular with new developers, and left the decision on which tasks and how much to use PP to the developers or to the teams. Pre-allocated sessions of 2–3 hours were recommended. The use of PP was voluntary.

The developers had positive attitudes to PP. Interestingly, attitudes improved further during the study, and at the end were as good as the attitudes to solo programming. However, the amount of PP increased slowly. The majority of developers continuously reported wanting to use more PP, but inadequate attention to resourcing PP, disturbing noise generated by PP, and inconvenient work spaces for PP were hindering an increase in its use. About a year after starting the adoption, a dedicated PP room was instituted. It provided both the needed space and the technical infrastructure to support PP, and helped shield other developers from the noise created by the pair programmers. During the latter half of the study, PP was used in ca. 10% of development work. [29]

3.3. Data collection

The data was collected using four surveys containing both open and closed question, and by discussing the results with the employees of the case department. In addition, the first author observed three PP sessions. The surveys, done between 3/2005 and 11/2006, were part of a larger case study on PP [29] and also covered areas not discussed in this paper. The head of the department sent the questionnaires by e-mail to all developers. Respondents returned the answers by e-mail to the first author. The identities of the respondents were not revealed to the company. Answering was voluntary. Most of the developers answered at least once, and the total number of different respondents was twenty-eight.

The main question was: **"***How does PP affect the following topics compared to solo programming? Answer based on your own PP experiences in this company.*" We listed eighteen aspects based on the effects proposed in the literature. The effect of PP on each aspect was evaluated on a 7-point scale ("7-higher", "4-no effect", "1-lower"). A higher value meant the desired effect for all but three aspects that were defect count, task effort and the exhaustiveness of work.

The teams did a mandatory code review for each implemented feature. In order to collect objective data on quality, we asked the teams to report the number and criticality of defects found in the reviews together with other feature attributes such as number of changed lines of code, effort, complexity and authors.

3.4. Data analysis

The analysis of the perceived effects focused on the answers from the last survey, because they embodied the longest experience with PP. However, answers from the earlier surveys were used, if a respondent did not answer the last survey. In an experiment a considerable positive change in the effects of PP happened after about ten hours of working with a pair [4]. Therefore we removed six respondents from the statistical analysis, who reported at most 10 hours of PP during the study. Three of them were new employees, who had not yet had many opportunities to do PP. The rest were senior employees. We used the Wilcoxon signed rank test to analyze the statistical significance of the results by comparing the evaluations of the effects of PP to the neutral value 4 indicating "no effect".

The answers to the open questions were taken from all surveys and classified thematically. Some representative citations are included in the following chapters.

4. Practicalities of pair programming

This chapter discusses how PP was used in the case organization, because the context may affect the effects of PP.

4.1. Application targets for pair programming

We asked the developers to evaluate how much PP they thought should be used for each activity of a task, if it had been decided to use PP for that task. This detail is interesting since the partners seldom performed a task completely using PP. Instead, the person responsible for a task also worked individually, a fact that may affect the magnitude of the effects of PP. We asked about the suggested usage of PP in the context of two goals: 1) improving quality and 2) teaching a junior. In the case department, each task (i.e. typically developing a feature) contained the same development activities (Table 2). In both cases the developers considered PP most useful for work breakdown structure (WBS), specification and heuristic analysis activities, whereas coding and module testing scored lower. For improving quality, there were clear differences in the averages between the activities, but for teaching the averages were higher, and their variance smaller.

A large proportion of the developers' comments proposed using PP especially for complex tasks. Software specification, challenging coding and problem solving were commonly mentioned examples.

"I think sw spec is the most critical activity, which should be done using PP in most tasks, and coding in some tasks. Usually the tasks that need PP more are quite complex, not necessarily big in the amount of changed lines, but when a lot of small changes are needed in the middle of the old code."

One developer considered testing and quite surprisingly, refactoring, which is typically quite difficult work, to be less suitable targets for PP.

"Probably testing is where PP should be used the least. Also refactoring would probably be more efficient when done alone."

Testing scored the lowest values in the quantitative evaluation (Table 2), although a few developers did explicitly mention testing and refactoring as good targets for PP. The general opinion was in contrast to [4], where 90% of testing and 70% of coding was done using PP.

Table 2. Proposed amount of pair programming for different activities (averages, N=15)

Goal	WBS	Specification	Heuristic analysis	Coding	Testing
Quality	60%	70%	80%	50%	40%
Teaching	80%	80%	80%	70%	60%

4.2. Pair programming sessions

Typically, the partners agreed in advance on arranging a PP session and reserved a couple of hours for it. Most developers considered 1.5–4 hours a suitable session length. Shorter sessions were considered inefficient, because it takes a while to get to speed, and in longer sessions, concentration decreases due to the exhaustiveness of PP. A developer mentioned the lack of breaks as a reason for the exhaustiveness. It may be that when working intensively with a pair you actually need more breaks than when working alone, but in practice you end up taking fewer.

"Not too long [sessions], because when working with a pair I spend more energy and have no one/two minute breaks, which seems to be important in IT-work."

"Longer sessions can be mentally exhausting, shorter sessions don't give time to really get the work started. Taking a break during longer sessions is recommended."

Between the sessions, the person responsible for the task typically continued on it alone. Sometimes the other person wanted to continue with the task. A technical problem was identified with sharing the code:

"Source code is always checked out to one of the pair programmers, which is a problem sometimes when the other wants to continue working without [the other person] checking all changes in the version control system."

4.3. Role switching

About half of the developers did not consider it useful to switch roles during a session, but they may switch them when starting the next session. The other half switched roles even several times per hour.

"[I switch] at least twice an hour. If both don't handle the keyboard, the other may not profit from PP. The less experienced partner should hold the keyboard more often than the other."

"Usually we do not switch keyboard during the sessions, but we switch coder/writer between different sessions."

Those who switched roles during sessions often considered it a good practice without reporting any drawbacks. It may be that those who did not switch roles had a preconception of getting nothing from it and have not even tried it. Our observation of some PP sessions where roles were not switched at all supports this. Even in these sessions, both partners were very active. All developers did not have a similar development environment, e.g. the same text editor, which was one reason for the reluctance to switching roles.

There can also be other responsibilities for the navigator, as revealed by a developer:

"The most important aspect has been that the other is steering (and most important of all: prioritizing) and assisting the other continuously (to the extent of supplying food and drinks!), while the other is deeper in implementing the solution."

4.4. Partner combinations

The opinions on good partner combinations were quite similar amongst the respondents. There should be a senior and a junior developer, or the partners should have knowledge areas that complement each other for the task at hand. However, a developer proposed random pairs, and avoiding bad combinations later if they occur. An answer mentioned that pairing two persons with very strong opinions may be a bad combination.

"In military terms, the pair could work optimally as a fighter pair, with a senior and a junior member. But the senior must not be a colonel and the junior a private soldier."

"PP can be very efficient when the pair is composed of people with different skill/knowledge sets that support each other."

5. Effects of pair programming

This chapter presents the results about the effects of PP compared to solo programming. The box plots illustrate the distribution of the answers for each aspect showing the maximum, quartile 3, median, quartile 1, and minimum values. Based on the Wilcoxon signed rank test the difference between PP and solo programming was significant ($p<0.05$) for all aspects except the exhaustiveness of work and estimation accuracy.

5.1. Learning

The respondents evaluated five aspects of learning: developed software, development tools, work practices, refactoring old code, and new technologies & programming languages (Figure 1). The answers were very positive with no answers on the negative side. The median was 6.0 for four aspects, and 5.5 for the last one.

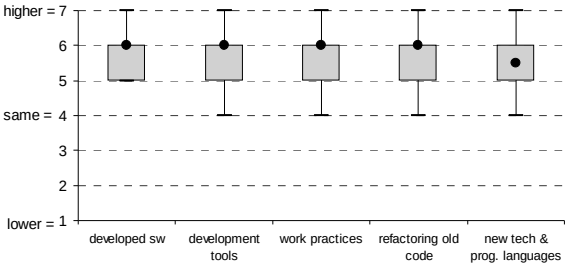

Figure 1. Effects on learning (N=22)

Additionally, we asked the developers to evaluate their familiarity with various product areas and work practices. Assuming that PP increases learning, the familiarity should increase more for those who use PP during the longitudinal study. However, there was no correlation between the amount of PP and the levels of familiarity, even though there was a clear positive trend in the median of the familiarity for most topics. Considering that nobody used PP for a large proportion of their work, and many other factors also contribute to learning, only a very large effect of PP could have been seen in this kind of analysis.

The very positive perceptions of learning may be explained by the fact that PP was used a lot for teaching junior developers. In that context improving learning is probably the most important goal for PP.

5.2. Quality

The respondents evaluated four aspects of quality: understandability and maintainability of code, defect count, and customer satisfaction (Figure 2). The effects on the understandability and maintainability of code were both positive to the same degree. The medians were 5.0 and there were no answers on the negative side. The opinions on the defect count indicated a slightly lower defect count when using PP. The median was 3.0 and there were no answers proposing an increase in the defect count. The effect on the customer/requirements management team's (RMT) satisfaction was smaller than for the other quality aspects, but the median 4.5 was on the positive side, and there were no answers on the negative side. The perceived effects were positive on all studied quality aspects, but smaller than for learning.

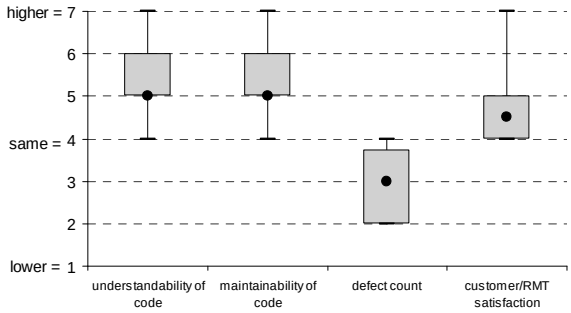

Figure 2. Effects on quality (N=22)

In addition, we analyzed defect data from the code reviews of each implemented feature. We compared the features developed using PP to those developed using solo programming. We failed to find any correlation between the defect count and the type of programming. However, we did not get data on factors that are likely to significantly affect the number of detected defects, such as preparation time for and the length of the reviews. In addition to these factors, the size, type and complexity of the features, and the proportions of old vs. new code varied a lot. For example, the inspection speed varied at least by a factor of 5, and the preparation time from zero to several hours. We think that the lack of data on these potentially significant factors hide the possible effect of PP on defect counts in our small data set.

5.3. Effort and schedule

The respondents evaluated the effects of PP on three aspects related to effort and schedule (Figure 3). The opinions on the effect on task effort varied a lot between the respondents, with answers distributing on both sides of neutral. This might suggest that there are some context factors that affect the effects, and the effects should be evaluated separately for different contexts. For example, a senior and a junior may experience opposite effects for the same task, if they compare the realized effort to what the task would have required from them alone. In addition, the type and complexity of work may affect the utility of PP. Finally, the differences in the amount of PP used for a task may affect the effect on effort. The median of the opinions was 5.0 indicating that in general PP takes somewhat more effort than solo programming, but the effect is very context specific.

The opinions on the effect on estimation accuracy varied quite a lot. The median of 4.5 indicated an almost negligible increase. According to some developers, estimating PP tasks was more difficult because PP is a less familiar way of working.

Even though estimation accuracy did not improve much, the opinions on the effect on the probability of finishing a task on schedule were very positive, with the median being 6.0. This benefit realized, e.g., because there were two persons who were able to contribute to a task, which helped if the other was too busy with other tasks or absent from work.

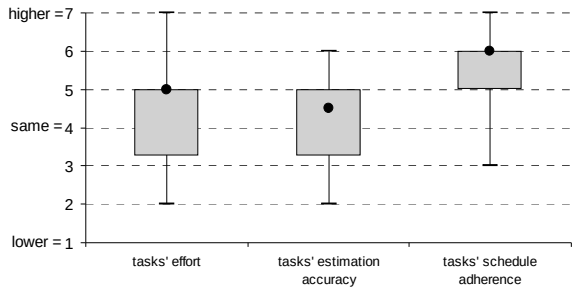

Figure 3. Effects on effort and schedule (N=22)

5.4. Human factors

The respondents evaluated the effects of PP on six aspects categorized here as human factors (Figure 4). PP had a large positive effect (median 6.0) on getting to know other developers. This is natural, because PP forces people to work with each other. Getting to know other developers better may also explain the positive effect on team spirit (median 5.5). The effect on the enjoyment of work was also positive, with the median being 5.0. A developer explained the effect on work enjoyment in more detail:

"PP decreases the amount of work related stress and the feeling of being irreplaceable because there are more people responsible for a piece of code."

All evaluations of getting familiar with others, team spirit and enjoyment of work were on the positive side, but the developers highlighted some considerations that should be kept in mind:

"Even though PP has almost every time been enjoyable and it raises the overall enjoyment of work, I still think that the best 'feeling of success' moments are from solo programming when I have solved some really big challenge."

"PP may have a positive effect on team spirit and enjoyment of work if used reasonably. Too much will have a negative effect."

The developers were supposed to follow the organization's implementation guidelines. The opinions on the discipline for following the agreed work practices indicated some increase, with the median being 5.0, and there were no negative answers.

In section 5.1 we saw that learning about refactoring old code improved a lot when doing PP. However, the amount of refactoring actually performed did not change (median 4.0). It may be that if refactoring is not an explicit part of a task, there was higher discipline also in following the unwritten guideline of not touching the old code in order not to break it.

The opinions on the effects on the exhaustiveness of work varied a lot, distributing to both sides of neutral (median 4.0). Based on the literature we expected some increase in exhaustiveness, and it realized in the first survey when the median was 5.0. However, the opinions changed after the respondents had used more PP. A possible explanation is that after you get used to work with a pair and become more familiar with the team you don't take so much stress from the PP sessions. Because of the exhaustiveness of PP there is probably some maximum amount of PP after which it is no more productive as commented by a developer:

"Even though PP reduces the overall burden of getting things done alone, it is usually quite tiresome. I would perhaps not be ready to make all development by PP voluntarily."

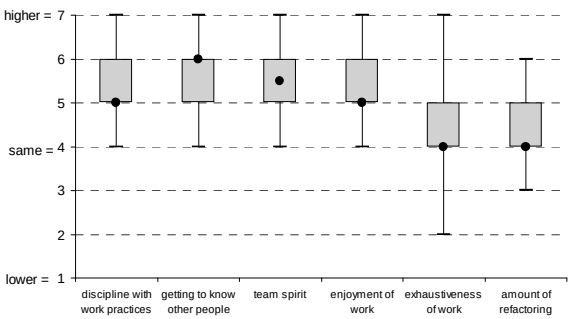

Figure 4. Effects on human factors (N=22)

6. Discussion

In this chapter we compare our results to those found in earlier studies, draw conclusions and discuss the limitations of our study.

6.1. Comparison to earlier studies

Most earlier studies have used students [4,5,6,9], or professionals doing small tasks [3,10,11] as subjects. The perceived effects in our study are similar to the effects of PP found in the earlier studies. The only exceptions were an almost negligible improvement in estimation accuracy proposed in [6], and the lack of increase in refactoring proposed in [1,20]. On average, our results support the proposed higher task effort of PP [3,4,5,6,7,10], but the variance was the highest of all aspects, with some respondents contradicting the proposition. The other proposed disadvantage, i.e. the higher exhaustiveness of work [1,8], was also perceived in the beginning of the study, but disappeared in the later surveys. This was the only aspect for which we were able to identify any clear trend between the surveys.

Two earlier studies from the industry [13,18] report quantitative data about large, −40% and −99.9%, decreases in the defect count. In our case, the decrease in defect count was also perceived but it was certainly smaller than the above figures. The developers' positive comments reported in [13] about schedule adherence, learning, team spirit, and various quality aspects were very similar to those found in our case.

6.2. Conclusions

The main contribution of our study is reporting rigorously collected data on the perceived effects of PP from a large scale, industrial software development context. Our study indicates that the effects of PP found in studies done in more limited contexts seem to be generalizable to realistic, industrial contexts.

We found positive effects on learning, quality, schedule adherence of tasks, getting to know other developers, team spirit, enjoyment of work, and discipline in following work practices. The perceived effect on task effort indicated a somewhat higher effort when using PP as opposed to solo programming. However, other benefits of PP might compensate for this, making the additional task effort acceptable from the perspective of overall productivity.

The perceived exhaustiveness of work was clearly higher for PP in the first survey, but it decreased to the level of solo programming in the last survey. However, there were still many individuals who considered PP more exhaustive than solo programming and vice versa. These individual preferences should be taken into account when planning how much PP is used and by whom.

The research community still needs more industrial data, both subjective and objective, on the effects of PP. In further studies, the context factors that may affect the effects of PP should be taken into account even more carefully, because e.g. in our study the perceptions of the effect on task effort were contradictory among the respondents.

6.3. Limitations

Our results are based on the perceived effects of PP instead of objective metrics. In an ideal situation, both subjective and objective data should be analyzed. Unfortunately, we were not able to institute the data collection procedures initially agreed upon with the case organization, resulting in a failure in getting the planned objective data. However, we believe that subjective data also has value, in particular since the effects of PP are multifaceted and thus hard to measure comprehensively. For example, a defect count metric involves numerous other attributes than just the number of defects. These include severity, type, detection effort and fixing effort for each defect. The significance of each of these attributes on the overall quality depends on the context. A human evaluator familiar with the context can give a more reliable evaluation of the overall effect of PP on the defect count than a set of metrics. The role of human evaluation increases if the effects of PP are inconsistent between the aspects, e.g., if certain types of defects decrease and some increase.

The effects of PP may change after it use is learned. Excluding the respondents, who had only little experience, should ensure that our results are based on a reasonable amount of PP experience.

The reliability of the survey instrument was analyzed by comparing the responses between 3rd and 4th survey for those persons who had not used much PP in between. There were little changes in their answers about the effects of PP indicating good reliability of the instrument.

References

[1] L. Williams and R. Kessler, *Pair Programming Illuminated*, Addison-Wesley, Boston, 2002.

[2] M. Cusumano, A. MacCormack, C.F. Kemerer, and B. Crandall, "Software Development Worldwide: The State of the Practice", *IEEE Software*, 20(6), 2003, pp. 28–34.

[3] J. Nosek, "The Case for Collaborative Programming", *Communications of the ACM*, 41(3), 1998, pp. 105–108.

[4] L. Williams, *The Collaborative Software Process*, Ph.D. dissertation, University of Utah, 2000.

[5] J. Vanhanen and C. Lassenius, "Effects of Pair Programming at the Development Team Level: An Experiment", *In Proceedings of International Symposium of Empirical Software Engineering (ISESE2005)*, 2005.

[6] J. Nawrocki and A. Wojciechowski, "Experimental Evaluation of Pair Programming", *In Proceedings of the 12th European Software Control and Metrics Conference*, 2001, pp. 269–276.

[7] K. Lui and K. Chan, "Pair programming productivity: Novice–novice vs. expert–expert", *International Journal of Human-Computer Studies*, 64(9), 2006, pp. 915–925.

[8] R. Gittins, S. Hope, and I. Williams, "Qualitative Studies of XP in a Medium Sized Business", *In Proceedings of the XP 2001 Conference*, 2001.

[9] M. Ciolkowski and M. Schlemmer, "Experiences with a Case Study on Pair Programming", *In Workshop on Empirical Studies in Software Engineering*, 2002.

[10] E. Arisholm, H. Gallis, T. Dybå, and D. Sjøberg, "Evaluating Pair Programming with Respect to System Complexity and Programmer Expertise", *IEEE Transactions on Software Engineering*, 33(2), 2007, pp. 65–86.

[11] M. Rostaher and M. Hericko, "Tracking Test First Pair Programming – An Experiment", *In Proceedings of XP/Agile Universe 2002*, 2002, pp. 174–184.

[12] H. Gallis, E. Arisholm, and T. Dybå, "An Initial Framework for Research on Pair Programming", *In Proceedings of International Symposium of Empirical Software Engineering (ISESE2003)*, 2003.

[13] A. Pandey, N. Kameli, and A. Eapen, "Application of Tightly Coupled Engineering Team for Development of Test Automation Software – A Real World Experience", *In Proceedings of the 27th Annual International Computer Software and Applications Conference (COMPSAC'03)*, 2003.

[14] G. Luck, "Subclassing XP: Breaking its rules the right way", *In Proceedings of the Agile Development Conference (ADC'04)*, 2004.

[15] B. Greene, "Agile Methods Applied to Embedded Firmware Development", *In Proceedings of the Agile Development Conference (ADC'04)*, 2004.

[16] L. Williams, "Extreme Programming Practices: What's on Top?", *Agile Project Management, Executive Report*, 12(5), Cutter Consortium, 2004.

[17] A. Belshee, "Promiscuous Pairing and Beginner's Mind: Embrace Inexperience", *In Proceedings of the Agile Development Conference (ADC'05)*, 2005.

[18] R. Jensen, "A Pair Programming Experience", *CrossTalk*, 16(3), 2003, pp. 22–24.

[19] L. Constantine, *Constantine on Peopleware*, New Jersey: Prentice Hall P T R, 1995.

[20] K. Beck, *Extreme Programming Explained*, Addison-Wesley, 2000.

[21] A. Van Deursen, "Program Comprehension Risks and Opportunities in Extreme Programming", *In Proceedings of the Eighth Working Conference on Reverse Engineering*, 2001, pp. 176–185.

[22] M. Müller, "Two controlled experiments concerning the comparison of pair programming to peer review", *Journal of Systems and Software*, 78(2), 2005, pp. 166–179.

[23] J. Wilson, N. Hoskin, and J. Nosek, "The Benefits of Collaboration for Student Programmers", *In Proceedings of the 24th SIGCSE Technical Symposium on Computer Science Education*, 1993, pp. 160–164.

[24] J. Chong, "Social Behaviors on XP and non-XP teams: A Comparative Study", *In Proceedings of the Agile Development Conference (ADC'05)*, 2005.

[25] A. Dick and B. Zarnett, "Paired Programming & Personality Traits", *In Proceedings of the XP 2002 Conference*, 2002.

[26] H. Sharp and H. Robinson, "An Ethnographic Study of XP Practice", *Empirical Software Engineering,* 9(1–2), 2004, pp. 353–375.

[27] S. Bryant, "Double trouble: Mixing qualitative and quantitative methods in the study of eXtreme Programmers", *In Proceedings of the 2004 IEEE Symposium on Visual Languages and Human Centric Computing*, 2004, pp. 55–61.

[28] M. Ally, F. Darroch, and M. Toleman, "A Framework for Understanding the Factors Influencing Pair Programming success", *In Proceedings of the XP 2005 Conference*, 2005.

[29] J. Vanhanen, C. Lassenius, and M. V. Mäntylä, "Issues and Tactics when Adopting Pair Programming: A Longitudinal Case Study", *In Proceedings of the Second International Conference on Software Engineering Advances (ICSEA 2007),* forthcoming.