Better, Not More Expensive, Faster? The Perceived Effects of Pair Programming in Survey Data

David Parsons
Hokyoung Ryu
Ramesh Lal
Institute of Information and Mathematical Sciences
Massey University
Auckland, New Zealand
Email: d.p.parsons@massey.ac.nz, h.ryu@massey.ac.nz, r.lal@massey.ac.nz

Abstract

There are many different techniques used with agile software development methods. Some of these, such as common coding guidelines and test driven development, are widely adopted and there appears to be a consensus that they can be beneficial. Others, however, are more controversial, none more so perhaps than pair programming. This technique meets resistance both from developers, who do not always wish to program with another person, and from managers, who see the sharing of a workstation as a potential barrier to programmer productivity. Its supporters, however, claim that it can have many benefits, in particular improving software quality. In this paper we look at the outcomes of previous research into the effects of pair programming and analyse some survey data to see how practitioners perceive its potential benefits for project outcomes in terms of quality, productivity, stakeholder satisfaction and cost. We conclude that the survey data appears to reinforce many of the previous claims made for the benefits of pair programming, but also raises questions that need further investigation.

Keywords

pair programming, eXtreme Programming, agile methods, survey

INTRODUCTION

Perhaps the most controversial practice advocated by some agile methods, in particular eXtreme Programming (XP), is pair programming. The primary aim of pair programming is to raise the quality of the software product by enabling a process of continuous code inspection (Beck 2005). However it is controversial because it is seen by some to have an immediately detrimental effect on programmer productivity. In addition there are human factors, such as social dynamics, lack of privacy, lack of 'quiet thinking time' and ergonomic issues (Stephens and Rosenberg 2003). Like refactoring, which may also appear to have a negative impact on productivity (Fowler 1999), pair programming is a practice that requires both faith and investment.

The intention of this paper is to explore some previous research into the outcomes from pair programming and to summarise the various claims made by researchers in this field. We then examine some survey data gathered by Ambler (2006) to see what it can tell us about how pair programming is perceived by software development practitioners. This paper therefore analyses a subset of the original data that relates specifically to pair programming. Using this data it explores some important questions about the extent to which pair programming and complementary practices are used in software development and how the impact of this practice is perceived.

In the next section we look at some previous research into pair programming and identify issues relating to this practice. We then introduce the data set that we have used in this study. This is followed by an analysis of the data and exposure of our findings and resulting issues. Finally, we provide some conclusions and propose some further work.

PAIR PROGRAMMING

The basic concept which drives pair programming is that two software developers, where one assumes the role of *driver* and the other of *navigator*, take part in a joint programming effort at one workstation. This combined effort, which requires frequent role changes between the two developers, and changes of pairing, has been argued to provide benefits such as improved team discipline, cohesion and morale, better code, a more resilient work flow and creation of better solutions. It raises the working knowledge of the entire code base by all the developers, and enables mentoring of team members. However, sitting two developers at a single keyboard is anathema to some managers, who see this as an instant halving of productivity. In this section we review some

of the key literature on the benefits and drawbacks of pair programming to see if there is any consensus on these issues.

What are the benefits of pair programming?

Early proponents of pair programming suggested that it may be a 'journey to greatness' (Williams and Kessler, 2000,) but such conclusions were based on what was then a limited set of evidence that was to large extent anecdotal. Subsequently, however, a great deal of effort has been invested in research into pair programming, utilising a range of methodologies including practical experiments in both academic and industrial settings. Various authors have suggested that pair programming will lead to an improvement in software quality (DeClue 2003; Hanks et al. 2004) resulting in end user satisfaction and a good return on investment in IT products. In addition, it can also lead to improvement in the satisfaction and morale of software team members (Mendes Al-Fakhri and Luxton-Reilly 2005), since individuals working in pairs often find it more enjoyable than going solo. A study by McDowell et al. (2003) in an academic context suggests that pair programming can be used as a tool to bolster the pass rate for programming courses and ensure course completion. According to Canfora et al. (2004), working in pairs not only enables individuals to build more knowledge, but also makes that knowledge building more stable than when working alone. McDowell et al. (2003) and Layman (2006) reveal that pairing with another student to design and code provides the following useful combined experiences; feeling accountable for the partner's achievements, being more organized, having advice and other opinions on solutions to difficult technical problems, getting tasks completed in a shorter period of time, avoiding coding errors and helping to identify errors in one's thinking when explaining ideas to the pair programmer. Freudenberg et al. (2007) claim that the major contribution of pair programming is in providing a context within which the level of talk necessary for collaborative software development can be framed. One of their findings was that talk about coding itself was relatively limited and that conversion was at a higher level of abstraction. In fact it is a significant finding that the conversation is at the same level of abstraction for both members of the pair, an observation also made by Salinger et al. (2008). Lui, Chan and Nosek (2008) seem to reinforce this assessment of the level of abstraction that pair programmers work at, with their conclusion that the benefits of pairing are seen primarily at the level of design rather than coding.

If the various claims made for the effects of pair programming are indeed true, then pair programming may lead to improvements not only in the quality of the software being implemented but also to the productivity of the programmers. However it is difficult to find compulsive evidence about measurable benefits. Experiments by Muller (2005), Hulkko and Abrahamsson (2005) and Arisholm et al. (2007) have produced results that are, at best, open to interpretation about the potential befits of pair programming in, for example, reducing the overall defect rate, or coping with complexity.

Does pair programming cost more?

If the benefits of pair programming are debatable, what does research tell us about its drawbacks? Even if pair programming can increase quality and satisfaction, some believe that there is a penalty to pay in terms of productivity and cost. Research relating to pair programming in an academic context by Williams et al. (2000) and Xu and Chen (2005) supports the claim that pairs complete their tasks faster than an individual developer, potentially resulting in quicker delivery of the software product. However these increases in overall speed may be small and do not take into account the increase in the overall amount of effort expended in total, which may range between 43% and 111% (Arisholm et al. 2007). The available research does not give much clarity to that debate. Xu and Chen (2005) state that pair programming increases the cost of development, but Williams et al. (2000) refute that claim, suggesting that the teamwork aspect mitigates some of the complex and lengthy compile and test activities involved in delivering working software, while helping to provide solutions to technical problems. Cockburn and Williams (2000) claim that there is only a 15% increase in overall development costs, whereas a study involving 15 full-time programmers by Nosek (1998) claimed that the cost could increase by about 42%. Another pair programming experiment involving students indicated that the cost could go up by 100% (Nawrocki and Wojciechowski 2001). However, Lui and Chan (2003), working with professional developers in an experimental context, concluded that this was a worst case scenario rather than the norm.

One aspect of the cost of pair programming is the initial investment required to introduce pair programming into a development team that has not previously used the practice. Again, previous research shows mixed results. Padburg and Muller (2004), while suggesting that pair programming suits those software development products where the product's time to market is critical, also downplay the potential start-up costs of moving to a pair programming approach. Their research indicates that the practice of pair programming is relatively easy to learn and use, provided that the individuals are motivated, committed, and believe that it will lead to major economic benefits while making the software development practice more comfortable and enjoyable. However, a study involving students by Vanhanen and Lassenius (2005) showed that pair programming teams had 29% lower

project productivity than teams comprising solo developers due to the time spent learning at the beginning of the project.

Limitations of previous studies

One aspect of previous studies into pair programming is that many of them have been undertaken in an academic context using students as the pair programmers. However, using pair programming with students is totally different when compared to using this practice in a real agile software development team, so the outcomes (quality, productivity, stakeholder satisfaction) will be different, and cost will be very difficult to estimate in a non-commercial context. Commercial agile teams are cohesive groups, where each individual member works for the team, and it is team success that counts at the end of the day. Team members know each others' skill sets and are able to pair with anyone else in the team at any time. Some teams swap pairs daily and it is the individuals who themselves decide whom they will pair for the day. Agile teams will have members who are expected to have excellent technical skills. However, it is their superior inter-personal skills (e.g. communication, interaction, ability to listen and digest another person's view, ability to handle critical remarks made by other team members on code and design issues, ability to help others and take up leadership roles) that are the main reasons why they are selected to work in an agile team. The hiring process for agile teams often involves putting a potential candidate for a day with the team to gauge his/her ability to work in that environment. In contrast, academia throws students together using an entirely different set of pairing criteria.

In an academic context, there are many constraints that will affect the relevance of results from pair programming experiments. Students in a class have varying technical and inter-personal skill sets. Who pairs with who would be critical for student performance, and who decides on a pair? Will a student want to work with a total stranger in their class? Will a team be made up of only two students? If they have problems with their tasks are they allowed to seek help from another team? How long it will take for them to bond with one another? Will pairs be co-located for the tasks? The student work environment and setup for pair programming is totally different from a professional development context. They may only be together for a few weeks, whereas with a professional agile team the individuals not only know one another but have worked with other team members for number of years. These are just some of the issues that make pair programming so different in academia when compared to a professional environment. This is an important consideration when drawing conclusions from the literature, due the preponderance of student-based studies on pair programming. For example, Dybå et al.'s (2007) analysis of relevant studies included 11 student based experiments out of a total of 15. Even where professional developers have been used for experiments (e.g. Liu and Chan 2003; Arisholm et al. 2007) we have to be aware that the environments in which the experiments took place were dissimilar to their normal working environments, where teams have evolved over a long periods of time. Artificially constructing teams for the purposes of measuring pair programming is therefore liable to distort the results. One exception to these studies is Vanhanen and Lassenius (2007), which reports the perceived effects of pair programming in a large scale, industrial software development context. This study reported improvements in quality but at the cost of greater overall effort required to implement system features.

Our investigation into the available literature on pair programming suggests that there appears to be general agreement that the main benefits of pair programming are probable (though not universally proven) improvements in product quality and an increase in satisfaction for individuals on the software development team. However, there is no general agreement about the impact of pair programming on the cost of software development to balance the claimed benefits of this practice. Furthermore, one cannot make specific claims about how it impacts the productivity of individuals or the satisfaction of stakeholders since the results from different studies vary widely.

THE AGILE ADOPTION SURVEY DATA

Most published research into pair programming is based on case studies or experiments, but surveys may provide us with a further opportunity for triangulation. A number of surveys on agile methods have been undertaken, but many of these have been administered by commercial organisations with small sample sizes and a limited range of questions, rarely addressing pair programming as a technique. However in this paper, we analyse parts of a data set gathered from an online survey with a large sample size that included the adoption of individual techniques, including pair programming, in its coverage. In the remainder of this paper, we introduce the data, show the results of our analyses and consider how our results relate to previous research findings.

The data set used in this paper was made available by Ambler (2006) and is based on an on-line survey that had 4,235 respondents. The respondents were self-selecting from the professional developer community based on the mailing lists of Dr. Dobbs Journal and Software Development Magazine. These magazines have readerships of 120,000 and 100,000 respectively, but there is a significant overlap between their two mailing lists. This readership, and thus the respondents, includes both agile and non-agile practitioners. Perhaps the most important aspect of the questionnaire is the four questions relating to the outcomes of software development projects,

namely; productivity, system quality, cost and stakeholder satisfaction. It also included questions about the agile techniques (including pair programming) that were being adopted, making it possible to see if certain practices and contextual factors could be correlated with certain outcomes. Amongst other questions, the survey asked; 'How have agile approaches affected your productivity?', 'How have agile approaches affected the quality of the systems produced?', 'How have agile approaches affected the cost of development?' and 'How have agile approaches affected the satisfaction of your business stakeholders in the work produced?' Each question was answered using a 5 point Likert scale with an additional 'Don't know' option.

In this paper, to explore the relationship between outcomes and the use of the pair programming technique, we focus on the outcomes from IT projects as dependent variables, with use of pair programming as the main independent variable, in association with the XP method and the related technique of co-location.

Of course there is a concern about the reliability of on-line survey data, which has not been controlled. Our statistical analyses of its results may be compromised by issues such as random errors, constant errors and the acquiescence effect. However we believe that a survey such as the one analysed here, which provides a large data set from a professional environment, and which does not confine itself only to the promoters of agile methods but to software developers using a range of methods, may nevertheless provide us with some useful insights. Another factor that may cause concern is that the survey is measuring perceived, rather then necessarily actual, effects on outcomes. However It should be noted that research in the literature that claims to measure actual effects has only been attempted in artificial conditions. Empirical work in the field, such as Vanhanen and Lassenius (2007), has of necessity measured perceived results since professional developers in their work environment cannot provide comparative data.

Observations from the data

We made some initial observations from the data related to the practice of pair programming and the contexts in which it was being used. Given that the practice of pair programming is most closely associated with the XP method in the literature, we were interested to see if this proved to be the case in practice. Table 1 shows the reported use of pair programming by respondents in different categories. From these figures it seems clear that a significant number of those who are using pair programming are also practicing XP.

From another perspective, however, the relationship between pair programming and XP appears weaker. Table 2 shows the numbers of respondents using pair programming, both with the XP method and with other methods, agile or not. We can confirm from these figures that pair programming is more popular within the XP method, but the figures are however rather low, with less than half of the respondents using XP claiming to use pair programming, regardless of whether XP is used in conjunction with other agile methods.

Table 1: Reported usage of pair programming by respondents

% using pair programming out of all respondents	13.82%
% using pair programming out of agile respondents	27.85%
% using pair programming out of non-agile respondents	3.12%
% using pair programming who are following XP	70.60%

Table 2: Reported usage of pair programming in different software development contexts

Pair Programming Context	Number	Sample size	% of sample
Using Pair Programming in any method	585	4,234	13.82%
Using Pair Programming but not within an agile method	75	2,403	3.12%
Using Pair Programming within one or more agile methods	510	1,831	27.85%
Using Pair Programming with XP combined with other methods	232	537	43.20%
Using Pair Programming with XP alone	181	416	43.50%
Using Pair Programming with other agile methods	97	885	10.96%

Adoption of pair programming by XP teams

Whilst there may be many reasons for the relatively low take-up of pair programming within XP and other agile methods, one possibility that could be considered is that the controversial nature of the technique has dissuaded developers from adopting this practice. However, if we look at the data further we can see that the take up of all

individual XP practices within the method is remarkably low. Not all of the XP techniques specified by Beck (2005) were included in the original survey. However it did include seven techniques that map closely to core practices of the XP method. These were; active stakeholder participation, code refactoring, code regression testing, co-location, continuous integration, pair programming and test driven design. Table 3 shows the number and percentage of respondents using each of these practices. The sample size for this table was 420, which was the number of respondents who claimed to be using XP and no other method. We excluded common coding guidelines from this table, even though it is an XP technique, because it is a highly generic technique that is applied across many development approaches, both agile and non-agile.

This data suggests that pair programming is not particularly out of line with several other XP techniques in terms of take-up. It is notable that code refactoring, which as we have indicated might also be regarded as one of XP's more controversial practices, has the highest adoption rate of these techniques. Equally important, the technique with the lowest take up was co-location, which is a fundamental support practice for pair programming, though it is possible that some pair programmers may be using tool support for distributed pair programming (Hanks 2004; Stotts et al. 2004).

Table 3: Actual use of seven core XP techniques among the sample who claimed to be following XP

Agile Technique used with XP	Number	Percentage of Sample
Active stakeholder participation	114	27.14%
Code refactoring	269	64.05%
Code regression testing	210	50.00%
Co-location	66	15.71%
Continuous integration	176	41.90%
Pair programming	183	43.57%
Test Driven Design (TDD)	180	42.86%

THE PERCEIVED IMPACT OF PAIR PROGRAMMING

From our review of the literature on pair programming, the data set provided by the Ambler survey, and some of our initial observations, we chose to address the following questions:

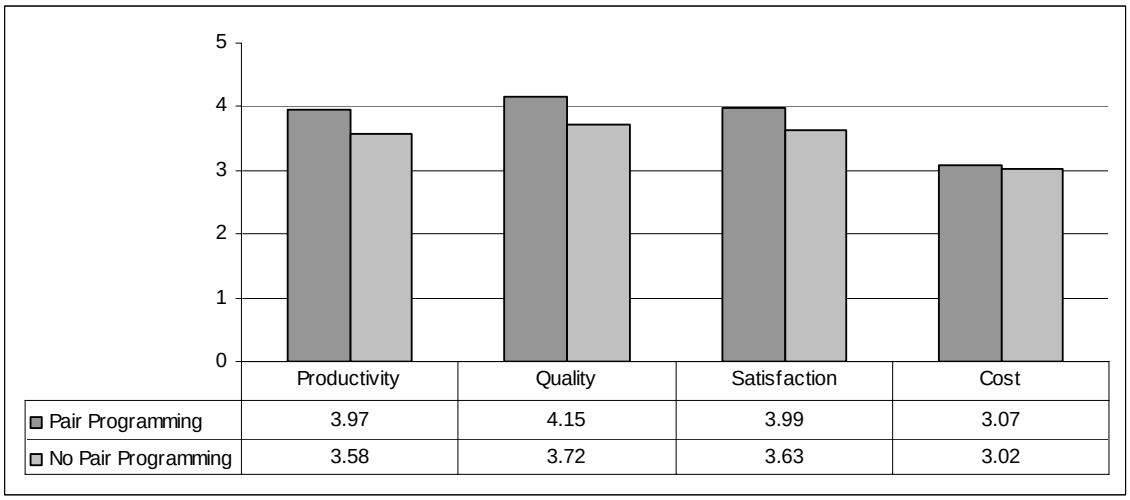
What is the perceived impact of pair programming practice on the four outcomes of quality, productivity, cost and satisfaction, in comparison with those who do not use this practice, across all development methods?

Given that pair programming is a core technique in eXtreme Programming, what is the perceived impact of pair programming practice on the four outcomes of quality, productivity, cost and satisfaction, in comparison with those who do not use this practice, amongst XP practitioners?

Given that many agile techniques are seen as synergistic when used in combination, are the perceived outcomes from using pair programming in XP affected by using it in conjunction with the closely related technique of co-location?

The Impact of Pair Programming on Project Outcomes

For our first analysis, we looked at the impact of the practice of pair programming on project outcomes across all software development methods, including those that did not use any agile methods. For each of the four outcomes, quality, productivity, cost and satisfaction, respondents who gave a 'don't know' response were excluded from the analyses below, so the sample size varies slightly for each outcome (i.e., 2,709 for productivity, 2,676 for quality, 2,593 for satisfaction, and 2,505 for cost.) For all of the data used in these analyses, respondents were asked to indicate their responses using a Likert scale from 1-5.



	Productivity	Quality	Satisfaction	Cost
Pair Programming	3.97	4.15	3.99	3.07
No Pair Programming	3.58	3.72	3.63	3.02

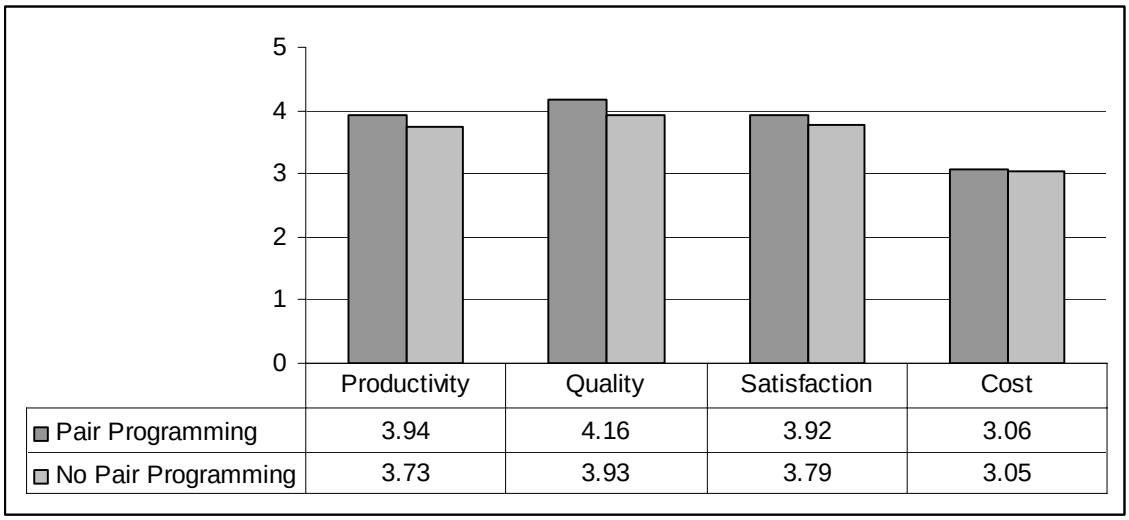
Figure 1. The impact of pair programming on the outcomes from software development projects

Figure 1 shows the comparison between reported outcomes for those respondents who used pair programming and those who did not. Relevant ANOVA (ANalysis Of VAriance) analyses revealed that the respondents claimed pair programming provided significantly better outcomes in terms of productivity ($F_{1,2708} = 104.24$, $p<.01$), quality ($F_{1,2675} = 133.06$, $p<.01$) and satisfaction ($F_{1,2592} = 87.77$, $p<.01$), without a significant influence on cost ($F_{1,2504} = 1.26$, n.s.)

Pair Programming in XP

For our next analysis we used data only from those respondents who claimed to be using XP as their only software development method. This might give us some indication if pair programming works better in a method that advocates its use as part of a set of techniques, or whether the perceived benefits we have reported across all methods are similarly reflected within an XP environment.

Our results are reported in Figure 2. In fact it appears that the reported benefits of using pair programming, though significant for both productivity and quality, are actually less within the practice of XP than when reported in use across all methods. It should be noted when analysing these results that it was not possible to perform ANOVA of the productivity data set (i.e., Levene's test for heterogeneity of variance was found to be significant, $F_{1,374} = 9.08$, $p<.01$), so non-parametric analyses (Mann-Whitney U tests) were consistently used instead for the complete data set in this example. They partially confirmed our previous interpretations, in that both productivity and quality were perceived as being improved, with no significant increase in cost. However there was no significant difference for satisfaction in this case. Because this analysis focused only on those respondents who claimed to be following only the XP method (and no other), the sample sizes, after removal of the 'don't know' responses, were much smaller than for the first analysis (376 for productivity, 372 for quality, 349 for cost and 361 for satisfaction).

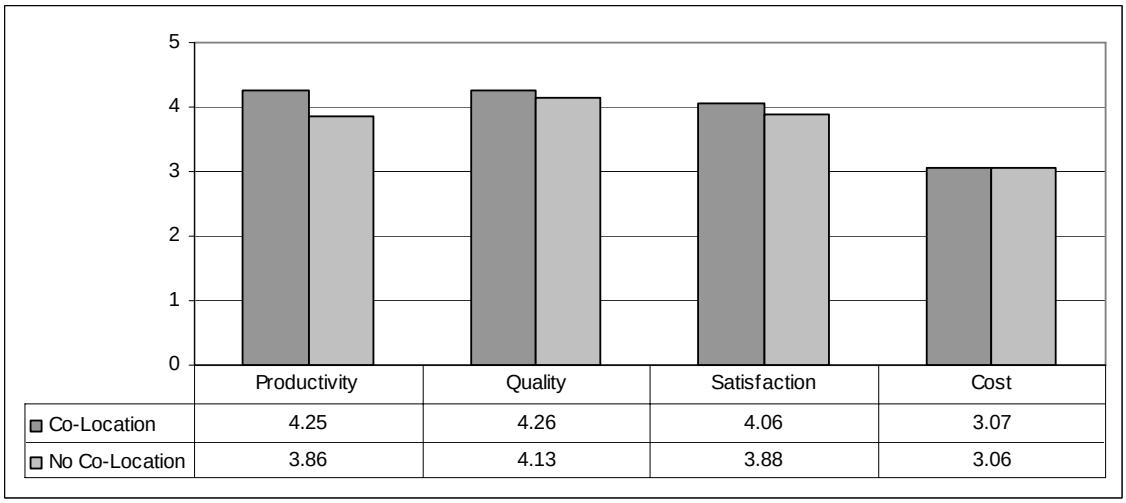


	Productivity	Quality	Satisfaction	Cost
Pair Programming	3.94	4.16	3.92	3.06
No Pair Programming	3.73	3.93	3.79	3.05

Figure 2. The impact of pair programming on the outcomes from eXtreme Programming projects

Pair programming in XP and the effects of co-location

For our final analysis, we looked at whether practitioners felt that the practice of pair programming had better results when used with the closely related practice of co-location. Since the number of pair programming practitioners in XP was significantly larger than the numbers using co-location, it would be useful to know if the discrepancy is likely to have impact on outcomes. In this analysis, we only considered those practitioners who were using pair programming; however, as mentioned above, relevant statistical analyses were only applied for the meaningful responses (i.e., the 'don't know' responses were excluded from the analyses). Consequently, the sample size was again different for each outcome.



	Productivity	Quality	Satisfaction	Cost
Co-Location	4.25	4.26	4.06	3.07
No Co-Location	3.86	4.13	3.88	3.06

Figure 3. The impact of supporting pair programming with co-location on the outcomes from XP projects

Our results are shown in Figure 3. In this analysis, reported productivity was shown to be significantly increased ($F_{1,172} = 7.83$, $p<.01$) when pair programming in the context of XP was used along with co-location. However, the other three outcomes (i.e., quality, satisfaction, and cost) did not seem to see significant effects from the closely related practice of co-location ($F_{1,168} = 0.79$, n.s.; $F_{1,166} = 1.17$, n.s.; $F_{1,155} = 0.01$, n.s; respectively).

DISCUSSION OF RESULTS

Our analysis of the perceived effects of pair programming on the four outcomes of productivity, quality, satisfaction and cost provided us with some interesting results. It appears that pair programming may be perceived to have a significantly beneficial effect on productivity, quality and satisfaction when measured as a technique across a range of different software development methods. However, when it is analysed within the

context of XP and again in conjunction with a very closely related practice (co-location) we find that the perceived benefits appear to be reduced. We may perhaps speculate that whilst pair programming can be a beneficial practice, the extent of its influence depends on the relative effects of other practices. XP for example is the most technique driven of the agile methods, and includes a number of different engineering practices. Many of these practices appear to synergise with each other so that we may reach a critical mass of techniques (Parsons et al. 2006). In this context, the relative impact of using a specific technique such as pair programming may be reduced, as other techniques may also be contributing to the overall perceived benefits. We can see from our analyses that the floor values for each set of results are increased in the three analyses (Table 4). This may contribute to a ceiling effect that sees a reduction of the significance of the effects of pair programming in each analysis.

Table 4: Increasing floor values for each of the three analyses

Measure	No pair programming	XP without pair programming	XP Pair programming, no co-location
Productivity	3.58	3.73	3.86
Quality	3.72	3.93	4.13
Satisfaction	3.63	3.79	3.88

CONCLUSION

In this paper we have reviewed some of the literature that relates to the debate about the benefits and potential drawbacks of the practice of pair programming. We have applied some statistical analyses to survey data that reports on project outcomes in terms of productivity, quality, cost and satisfaction, using pair programming as the main unit of analysis. Our results indicate that pair programming as a software development practice is perceived by practitioners as having significant benefits, without a significant increase in cost. This result appears to reinforce the more positive outcomes identified from previous research, but does not support the hypothesis that pair programming increases costs significantly. In the context of XP, it also appears that pair programming is perceived to make a significant contribution to project outcomes, but only in terms of productivity and quality. Further, respondents reported that using pair programming along with the closely associated practice of co-location appeared to lead to a significant increase only in productivity. We have proposed that this may be an effect of relativity, as a consequence of adopting multiple synergistic techniques, thus reducing the perceived impact of a single technique.

Of course this survey data can only be said to reflect the perceptions of a self selecting group of practitioners. However the results gleaned from our analysis may provide some insights into aspects of pair programming that are worth further investigation. For example, how much of the perceived benefits of pair programming are based on a self fulfilling prophecy affect rather than an objective measure of improvement in quality and better management of complexity? One survey cannot answer these questions, but nevertheless it raises some interesting issues that are worthy of further investigation, in particular the suggestion that perhaps it is productivity that can benefit most from the technique of pair programming, a somewhat counter intuitive result. However, even if we debate the validity of the apparent quantitative benefits described in this paper, we perhaps need to look more closely at the qualitative benefits that may accrue, over longer periods of time and at higher levels of system complexity.

REFERENCES

Ambler, S. 2006. “Agile Adoption Rate Survey,” Retrieved 16th September, 2008, from http://www.ambysoft.com/surveys/agileMarch2006.html

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

Beck, K. 2005. *Extreme Programming Explained: Embrace Change* (2nd Edition). Boston: Addison-Wesley.

Canfora, G., Cimitile, A., and Visaggio, C. 2004. “Working in Pairs as a Means for Design Knowledge Building: An Empirical Study,” *12th IEEE International Workshop on Program Comprehension*, Bari, Italy, June 24-26 2004.

Cockburn, A., and Williams, L. 2000. “The Costs and Benefits of Pair Programming,” *First International Conference on Extreme Programming and Flexible Processes in Software Engineering (XP 2000)*, Cagliari, Sardinia, Italy, June 2000.

DeClue, T. 2003. "Pair programming and pair trading: effects on learning and motivation in a CS2 course," *The Journal of Computing Sciences in Colleges*, (18:5) pp 49-56.

Dybå, T., Arisholm, E., Sjøberg, D., and Hannay, J. 2007. "Are Two Heads Better than One? On the Effectiveness of Pair Programming," *IEEE Software*, (24:6) pp 12-15.

Fowler, M. 1999. *Refactoring: Improving the Design of Existing Code*. Reading, Mass.: Addison-Wesley.

Freudenberg (née Bryant) S., Romero, P. and du Boulay, B., 2007. "'Talking the talk': Is intermediate-level conversation the key to the pair programming success story?" *Agile 2007* Washington, DC, 13-17 Aug. 2007, pp 84 – 91.

Hanks, B. 2004. "Distributed Pair Programming: An Empirical Study," *Extreme Programming and Agile Methods - XP/Agile Universe 2004 Conference*, Calgary, Canada, August 15-18, 2004

Hanks, B., McDowell, C., Draper, D. and Krnjajic, M. 2004. "Program quality with pair programming in CS!," *ACM SIGCSE Bulletin* (36:3) pp 176-180.

Hulkko, H., and Abrahamsson, P. 2005. "A multiple case study on the impact of pair programming on product quality," *27th international conference on software engineering*, St. Louis, MO, USA

Layman, L. 2006. "Changing student's perception: an analysis of supplementary benefits of collaborative software development," *19th Conference on Software Engineering Education & Training*, Oahu, Hawaii, April 19-21, 2006.

Lui, K., and Chan, K. 2003. "When Does a Pair Outperform Two Individuals?" *4th International Conference on Extreme Programming and Agile Processes in Software Engineering (XP 2003)*, Genova, Italy, May 25-29, 2003.

Lui, K., Chan, K. and Nosek, J. 2008. "The Effect of Pairs in Program Design Tasks," *IEEE Transactions on Software Engineering* (34:2) pp 197-211.

McDowell, C., Bullock, H., and Fernald, J. 2003. "The impact of pair programming on student performance, perception and persistence," *25th international conference on software engineering*, Portland, Oregon, USA, May 3-10 2003.

Mendes, E., AL-Fakhri, L., and Luxton-Reilly, A. 2005. "Investigating pair-programming in a 2nd-year software development and design computer science course," *10th annual SIGCSE conference on innovation and technology in computer science education*, Lisbon, Portugal.

Muller, M. 2005. "Two controlled experiments concerning the comparison of pair programming to peer review," *The Journal of Systems and Software* (78) pp 166-179.

Nawrocki, J., and Wojciechowski, A. 2001. "Experimental evaluation of pair programming," *12th European software control and metrics conference*, London, England, April 2001.

Nosek, J. 1998. "The case for collaborative programming," *Communications of the ACM*, (41:3) pp 105-108.

Padberg, F., and Muller, M. 2004. "Modeling the impact of a learning phase on the business value of a pair programming project," *11th Asia-Pacific software engineering conference*, Busan, Korea, December 2004.

Parsons, D., Ryu, H., and Lal R. 2007. "The impact of methods and techniques on outcomes from agile software development projects'" *IFIP TC8 Working Conference on Organizational Dynamics of Technology-Based Innovation* (McMaster, T., Wastell, D., Ferneley, E. & DeGross, J. Eds.), p. 235-249, Springer, NY.

Salinger, S., Plonka, L. and Prechelt L. 2008. "A Coding Scheme Development Methodology using Grounded Theory for Qualitative Analysis of Pair Programming." *Human Technology* (4 :1) pp 9-25.

Stephens, M., and Rosenberg, D. 2003. *Extreme Programming Refactored: The Case Against XP*. Berkley, CA: Apress.

Stotts, D., Smith, J., and Gyllstrom, K. 2004. "Support for Distributed Pair Programming in the Transparent Video Facetop," *Extreme Programming and Agile Methods - XP/Agile Universe 2004 conference*, Calgary, Canada, August 15-18, 2004.

Vanhanen, J., and Lassenius, C. 2005. "Effects of pair programming at the development team level: an experiment," *2005 International symposium on empirical software engineering*, 2005.

Vanhanen, J., and Lassenius, C. 2007. "Perceived Effects of Pair Programming in an Industrial Context," *33rd EUROMICRO Conference on Software Engineering and Advanced Applications*, Lubeck, 28-31 Aug. 2007, pp 211-218.

Williams, L., and Kessler, R. 2000. "All I really need to know about pair programming I learned in kindergarten," *Communications of the ACM* (43:5) pp 108-114.

Williams, L., Kessler, R., Cunningham, L., and Jeffries, R. 2000. "Strengthening the case for pair programming," *IEEE Software*, July/August 2000, pp 19-25.

Xu, S., and Chen, X. 2005. "Pair programming in software evolution," *Canadian conference on electrical and computer engineering*, Saskatoon, Canada, May 1-4, 2005.

COPYRIGHT