Editor: Forrest Shull ■ Fraunhofer Center for Experimental Software Engineering, Maryland ■ fshull@fc-md.umd.edu

Are Two Heads Better than One?

On the Effectiveness of Pair Programming

Tore Dybå, Erik Arisholm, Dag I.K. Sjøberg, Jo E. Hannay, and Forrest Shull

While working my way through graduate school in the mid-90s, I took every chance I had on summer and holiday breaks to go back to work as a software developer. Luckily, I worked for a great boss and a flexible organization, which could usefully put me back to work extending the applications I worked on during my previous visit. If there was a downside, it was that I usually got assigned to whichever random desk was open during each visit. All too often, I ended up at the dreaded solitary desk in the windowless server room, among the empty computer boxes and the walls of sheetrock and spackle.

These experiences (which are not so uncommon, I suspect, among junior software developers) are probably among the reasons why I've found pair programming (PP) so interesting. Many developers, and not just those who have ended up programming alone in windowless offices, have been excited by the paradigm shift, while others seem extremely annoyed by it. In both cases, perhaps the most important result is that PP leads to rethinking about the concept of development teams and about how individual programmers can best contribute to the project. Now that PP is several years old and has seen increasing interest and adoption, it's useful to consider what has been learned about its more specific effects. The evidence certainly provides proof of its benefit, although not in all cases and perhaps not in the contexts that many developers would have thought. —Forrest Shull

The familiar adages "two heads are better than one," "many hands make light work," and "too many cooks spoil the broth" suggest that working with others sometimes facilitates performance, yet sometimes hinders it. Pair programming is a collaborative approach that makes working in pairs rather than individually the primary work style for code development. Because PP is a radically different approach than many developers are used to, it can be hard to predict the effects when a team switches to PP. Luckily, we have evidence that takes a hard look at such effects. Because projects focus on different things, this article concentrates on understanding general aspects related to effectiveness, specifically project *duration* (the calendar time required to produce a given system), *effort* (the person-hours required), and *quality* (how good the final product is).

Advocates claim that, when applied to new code development or used to maintain and enhance existing code, PP has many benefits over individual programming, such as producing higher-quality code in about half the time.[1] (Other claimed benefits include happier programmers and improved teamwork, knowledge transfer, and learning. We don't address these benefits in this article except insofar as these factors might impact effort, duration, or quality.) Interestingly, these claims seem at first to contradict decades of research in psychology focused on understanding productivity in small groups; such studies typically have found that small groups usually are less productive than expected for many tasks.[2–4]

An important question, therefore, is whether empirical evidence substantiates the claims regarding PP and how PP relates to such group research. A good answer to this question could be helpful in making the right decisions about resource allocation and managing people and time.[5] To answer this, we systematically identified all the existing studies on PP, characterized the context in which each was run, and then examined whether the study results were consistent with one another. If they were, we had some confidence in the results; if not, we investigated the individual studies on each side of the issue to locate any patterns that could help us understand whether similar projects were showing similar results after all.

Studies on effectiveness

As developers have increasingly become interested in PP, the research community's interest has increased as well. In our systematic review of the literature, we found 15 studies that compared the effects of pair programming and of individual programming. We refer to these studies with codes of either "P" or "S" (signifying whether the study's subjects were professionals or students), followed by the year the study was published. The full list of studies (along with additional description of the size and context of each) is available at www.computer.org/software.

The first such study was conducted in 1998.[6] Although the research grew slowly in the following years, with roughly one controlled study per year in the early 2000s, recent years have seen a sharp increase. Three studies were published in 2005, four in 2006, and two as of August 2007.

There were 15 studies:

- Four were run with professionals and 11 with students.
- Ten were from Europe and five from North America.
- Eleven compared the effectiveness of isolated pairs versus isolated individuals, and only four studies made the comparison in a team context (that is, a context in which multiple individuals or pairs worked together to develop a software product).
- All studies used programming tasks as the basis for comparison.
- The number of subjects in the studies varied from 12 to 295, with a median of 24.

Across these studies, the attributes of interest were measured as follows:

- Duration was reported either as the total time a subject took to complete all the assigned programming tasks (that is, each subject judged for himself or herself when the task was done) or as the total time taken to produce code that was assessed as having reached a certain quality standard (that is, the researcher checked that the task products were acceptable).
- For comparisons between pairs and solo programmers, pair effort was reported as twice the duration of each individual in the pair. For team-based comparisons, such as teams of individuals versus teams of pairs, effort was reported as the total effort of all team members.
- Quality typically was reported as the number of test cases passed or number of correct solutions of programming tasks. However, some studies also used student grades, the amount of desired functionality delivered, and code metrics as measures of quality.

An important question is whether empirical evidence substantiates the claims regarding pair programming.

As is often the case in empirical software engineering, the studies we included in our analysis don't all apply the same measures and weren't all run in similar contexts. Rather, each investigated PP's effectiveness with respect to different aspects of at least one of the concepts of quality, duration, and effort. So, we couldn't simply combine the data and see whether the numbers show that pairs consistently did better or worse than individuals. Instead, we needed to conduct a meta-analysis in which we understood what each study showed but also treated each study as an individual piece of evidence. The objective had to be to see whether a certain weight of evidence exists that shows a consistent effect in different contexts, with different measures, and so forth. If an effect is truly robust, it should show up across multiple studies, not just in isolated instances.

To compare across studies, we had to standardize effect sizes. In this meta-analysis, we used the mean value seen for pairs minus the mean value seen for individuals, but standardized with respect to the standard deviation and corrected for small-sample bias.[7] This produced a metric (Hedges' *g*) that we could interpret consistently for all studies, regardless of how the studies had measured the concepts. An effect size of 0.5 thus indicates that the mean of the pairs is half a standard deviation larger than the mean of the individuals. So, if a study's effect size is 0, pairs and individuals performed roughly the same. A positive effect size means that pairs did better, on average, than individuals, and a negative measure indicates that individuals on average came out ahead. For studies in software engineering, effect sizes of 1.00–3.40 are large, effect sizes of 0.38–1.00 are medium, and effect sizes of 0–0.37 are small.[8]

Figure 1 uses a forest plot to graphically summarize these results, allowing easy assessment of whether the individual studies tend to agree with one another. The figure shows a separate analysis for each of the three outcome variables: quality (for which 11 studies provided appropriate data), duration (11 studies), and effort (7 studies). If a study has several indicators, the mean

	Study	Effect size	Lower limit	Upper limit	Relative weight
	P07a	0.11	−0.24	0.46	23.24
	S06a	0.08	−0.28	0.43	22.85
	S00	1.04	0.65	1.43	18.64
	S03	0.10	−0.44	0.64	9.80
	S05b	0.28	−0.32	0.88	7.95
	P07b	0.69	−0.09	1.46	4.73
	S02	0.30	−0.50	1.09	4.53
	S06c	0.32	−0.69	1.32	2.81
	S06b	0.51	−0.59	1.62	2.33
	P98	0.91	−0.28	2.10	2.02
	S06d	2.20	0.58	3.82	1.09
(a)	**Overall effect**	**0.38**	**0.21**	**0.55**	
	P07a	0.21	−0.13	0.54	33.33
	S05a	0.57	0.07	1.07	15.24
	S01	0.16	−0.40	0.73	11.83
	S03	0.55	−0.03	1.14	10.97
	S05b	1.30	0.64	1.95	8.66
	P07b	−0.59	−1.37	0.20	6.09
	S02	−0.05	−0.84	0.74	6.01
	P02	0.06	−1.08	1.21	2.87
	S06b	0.98	−0.24	2.20	2.51
	S06d	1.85	0.34	3.35	1.65
	P98	4.09	1.98	6.20	0.84
(b)	**Overall effect**	**0.40**	**0.21**	**0.59**	
	P07a	−0.68	−1.03	−0.34	47.66
	S05a	−1.09	−1.61	−0.58	21.17
	S05b	−0.25	−0.85	0.35	15.73
	S05c	−0.49	−1.62	0.63	4.46
	S06b	−0.64	−1.83	0.54	4.04
	S06d	−0.11	−1.31	1.10	3.88
	S06c	2.52	1.17	3.88	3.07
(c)	**Overall effect**	**−0.57**	**−0.81**	**−0.33**	

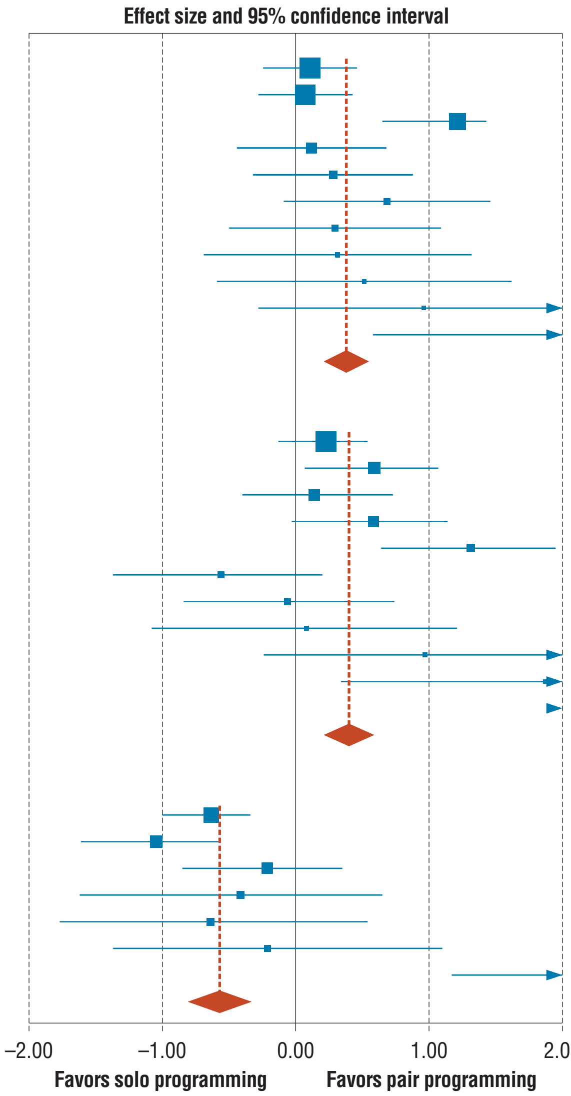

Figure 1. Meta-analyses of pair programming's effects on (a) quality, (b) duration, and (c) effort.

outcome is shown. For each study, the square box indicates the estimated effect size, and the length of the line to either side of the square represents the 95 percent confidence interval for that estimate. (Studies with less variance have smaller confidence intervals.) The bottom line in each analysis contains a plot with a diamond, which indicates the overall effect size for the meta-analysis. The line's center and width give the estimated mean and its 95 percent confidence interval of the population effect size, respectively.

Not all studies contribute equally to that overall conclusion. In each analysis, the studies are sorted by the relative weight that the study's effect size receives in the meta-analysis (represented by the size of each square). Each study's weight is inversely proportional to the variance. Estimates from larger studies will usually be more precise than those from smaller studies; so, larger studies will generally receive greater weight.

Regarding PP's effect on quality, the meta-analysis shows a general agreement among the studies and suggests that PP leads to a medium-sized increase in quality compared with individual programming (effect size = 0.38). All the included studies show that PP positively affects quality, although most of these effect sizes are small to medium.

Regarding duration, the meta-analysis suggests that PP has a medium-sized overall reduction of the time to deliver the finished product, compared with individual programming (effect size = 0.40). Unlike the analysis for quality, the studies on duration show a more mixed picture; two of the 11 studies show a negative effect, while the remaining nine show a positive effect.

Regarding effort, the meta-analysis suggests that there's a medium-sized negative effect due to PP when compared with individual programming (effect size = –0.57). All but one study show a negative effect on effort—that is, that working in pairs requires more person-hours to develop the same software. The one exception is study S06b.[9] However, the results of that study aren't directly comparable because this was the only study that compared teams of pairs with teams of individuals who also performed inspections in addition to implementing the software. (It's an interesting question whether this was actually the more fair comparison—that is, whether having an extra pair of eyes on the code in PP also delivers the benefits of code inspections at the same time the software is being developed.)

Task complexity and expertise

From the results we've reported here, we found general agreement that PP leads to increased quality. The picture is muddier concerning the effort and calendar time required to reach that increase. So, we next investigated whether other factors caused the differences among the studies. The relatively small overall effects and large differences in the reported effects between the individual studies indicated that one or more additional variables might have been playing a significant role. In particular, the psychological literature on group dynamics[10] shows that the extent to which group performance exceeds that of individuals depends on the group's composition and the tasks' characteristics. It therefore seemed important to investigate whether such effects could be influencing the studies' outcomes differently.

Unfortunately, only two of the studies that we found, S05c[11] and P07a,[12] tested explicitly for these effects. S05c found that task complexity didn't affect the differences in effort between solo and pair programming, but it didn't investigate group composition.

Study P07a addressed these factors more thoroughly and achieved a contradictory result. This study, in which 295 Java consultants were hired for a day, was the largest in our set. The study examined how effectively participants at three levels of expertise (junior, intermediate, and senior) made changes to systems with two distinct designs (one easier than the other). The results showed that by cooperating, programmers could complete tasks and attain goals that would be difficult or impossible if they worked individually. Yet, the higher quality for complex tasks came at a price of a considerably higher effort (cost), while the reduced completion time for the simpler tasks came at a price of a noticeably lower quality. Table 1 summarizes these relationships.

Table 1

Guidelines for when to use PP

Programmer expertise	Task complexity	Use PP?
Junior	Easy	Yes, provided that increased quality is the main goal
	Complex	Yes, provided that increased quality is the main goal
Intermediate	Easy	No
	Complex	Yes, provided that increased quality is the main goal
Senior	Easy	No
	Complex	No, unless you're sure that the task is too complex to be solved satisfactorily by an individual senior programmer

Not unexpectedly, our meta-analysis showed that the question of whether two heads are better than one isn't precise enough to be meaningful. Given the evidence, the best answer is "it depends"—on both the programmer's expertise and the complexity of the system and tasks to be solved. Two heads are better than one for achieving correctness on highly complex programming tasks. They might also have a time gain on simpler tasks.

Additional studies would be useful. For example, further investigation is clearly needed into the interaction of complexity and programmer experience and how they affect the appropriateness of a PP approach; our current understanding of this phenomenon rests chiefly on a single (although large) study. Only by understanding what makes pairs work and what makes them less efficient can we take steps to provide beneficial work conditions, to avoid detrimental conditions, and to avoid pairing altogether when conditions are detrimental. With the right cooks and the right combination of ingredients, the broth has the potential to be very good indeed.

References

1. L. Williams and R. Kessler, *Pair Programming Illuminated*, Addison-Wesley, 2002.
2. I.D. Steiner, *Group Process and Productivity*, Academic Press, 1972.
3. J.M. Levine and R.L. Moreland, "Progress in Small Group Research," *Ann. Rev. Psychology*, vol. 41, 1990, pp. 585–634.
4. N.L. Kerr and R.S. Tindale, "Group Performance and Decision Making," *Ann. Rev. Psychology*, vol. 55, 2004, pp. 623–655.
5. T. Dybå, B.A. Kitchenham, and M. Jørgensen, "Evidence-Based Software Engineering for Practitioners," *IEEE Software*, vol. 22, no. 1, 2005, pp. 58–65.
6. J.T. Nosek, "The Case for Collaborative Programming," *Comm. ACM*, vol. 41, no. 3, 1998, pp. 105–108.
7. M.W. Lipsey and D.B. Wilson, *Practical Meta-Analysis*, Sage, 2001.
8. V.B. Kampenes et al., "A Systematic Review of Effect Size in Software Engineering Experiments," to be published in *J. Information and Software Technology*, 2007.
9. M.M. Müller, "A Preliminary Study on the Impact of a Pair Design Phase on Pair Programming and Solo Programming," *J. Information and Software Technology*, vol. 48, no. 5, 2006, pp. 335–344.
10. D.R. Forsyth, *Group Dynamics*, 4th ed., Thomson Wadsworth, 2006.
11. J. Vanhanen and C. Lassenius, "Effects of Pair Programming at the Development Team Level: An Experiment," *Proc. Int'l Symp. Empirical Software Eng.* (ISESE 05), IEEE CS Press, 2005, pp. 336–345.
12. E. Arisholm et al., "Evaluating Pair Programming with Respect to System Complexity and Programmer Expertise," *IEEE Trans. Software Eng.*, vol. 33, no. 2, 2007, pp. 65–86.

Tore Dybå is a visiting researcher at the Simula Research Laboratory and a chief scientist at Sintef ICT. Contact him at tore.dyba@sintef.no.

Erik Arisholm is a project manager at the Simula Research Laboratory and an associate professor at the University of Oslo's Department of Informatics. Contact him at erika@simula.no.

Dag I.K. Sjøberg is a research director at the Simula Research Laboratory and a professor at the University of Oslo's Department of Informatics. Contact him at dagsj@simula.no.

Jo E. Hannay is a visiting researcher at the Simula Research Laboratory and an associate professor at the University of Oslo's Department of Informatics. Contact him at johannay@simula.no.

Forrest Shull is a senior scientist at the Fraunhofer Center for Experimental Software Engineering, Maryland, and director of its Measurement and Knowledge Management Division. Contact him at fshull@fc-md.umd.edu.