Pair Programming vs. Solo Programming: What Do We Know After 15 Years of Research?

Carolina Alves de Lima Salge  
University of Georgia  
csalge@uga.edu  

Nicholas Berente  
University of Georgia  
berente@uga.edu  

Abstract

In 2000, Kent Beck popularized the notion of pair programming, and argued that this practice could improve software quality and developer learning, and would not adversely affect duration, as compared to solo programming. We conduct a meta-analysis of 15 years of empirical tests of these relationships, and find that: pairs generally (a) produce higher quality software, (b) learn more, and (c) program faster, supporting Beck’s arguments. We conclude the paper with what we know and what we do not know about pair programming and directions for future research on the topic.

1. Introduction

Pair programming continues to be one of the most hotly contested practices of agile software development. Adherents swear by it and opponents object to it with equal vigor. Although the organizational practice of pair programming has been traced back as early as 1978,1 pair programming gained widespread popularity with the agile movement – following Kent Beck’s (2000) inclusion of pair programming as one of the eXtreme Programming (XP) approach’s fundamental practices. Since then we have had 15 years of research testing the claims of pair programming. In this paper we ask: what do we know after 15 years of research on pair programming?

To answer this question we conducted a quantitative meta-analysis of published studies. In this meta-analysis, we focused on three key areas where Beck argued that pair programming purportedly impacts software development: quality, learning, and duration. We conclude that pair programming does indeed appear to be positively related to each of these outcomes, based on existing studies. Essentially pair programming does generally improve software quality and learning of participants, and also appears to speed up software development.

We organize this paper as follows. First we root ourselves in Beck’s thinking when he popularized the notion of pair programming in 2000 with respect to the impact of pair programming on quality, learning, and duration. Then we present a meta-analysis testing these three outcomes of pair programming. Rooted in the results of the meta-analysis, we then discuss what we do know and what we do not know about pair programming now.

2. Beck’s Argument for Pair Programming

“Pair programming” describes the practice of two developers coding together using a single computer keyboard and mouse [4]. One coder is the “driver” and the other is the “navigator.” The driver actually performs the coding, whereas the navigator aids the driver in thinking through issues, searching for errors, discussing alternatives, etc. There are three broad areas where pair programming is expected to affect software development: quality, learning, and duration. To think through why this is the case, we will present Kent Beck’s original arguments, then discuss whether these arguments are supported by research.

First, Beck indicated that quality will improve due to pair programming for the fundamental reason that two people are more likely to be able to solve complex problems more effectively than just one, and a dialog between two developers is likely to generate better outcomes than either acting alone. But it is also an issue of control. As Beck indicated: “if people program solo they are more likely to make mistakes, more likely to overdesign, and more likely to blow off the other practices, particularly under pressure.” (p.56) Pair programming is a way to essentially subject coding activity to scrutiny and deliberation. Yes, two heads are better than one, but also two heads will not take shortcuts and will be less sloppy. Quality is therefore enhanced by the additional substantive problem solving capabilities an additional developer brings, and this is improved by the continuous communication.

1 http://guide.agilealliance.org/guide/pairing.html
between developers. Quality is also enhanced by the 
additional procedural rationalization resulting from the 
scrutiny of an additional developer.

Second, although it was not one of his central 
arguments, Beck indicated that pair programming can 
impact the learning outcomes of developers. He 
indicated that in participating in pair programming, 
developers learn from each other. This is particularly 
important when pairing junior with senior developers – 
[junior developers] learn more rapidly. Further, pair 
programming in XP is intended to be dynamic. 
Developers will pair up with different developers 
regularly, and this cross-pollination throughout 
different sets of pairs will improve the learning in the 
group. Finally, the constant communication during the 
development of complex software products will 
improve a variety of skills of developers, including 
interpersonal and soft skills. As Beck indicated: “You 
quickly learn to talk at many different levels.” (p.82) 
In sum, a variety of different forms of learning are 
thought to result from participation in pair 
programming.

Third, the main objection that Beck anticipated to 
pair programming is simply that “It will be too slow.” 
(p.56) There are multiple dimensions to this - clearly 
two people doing a job that one person can do, that’s 
twice the effort. But is it? What if the advantages of 
thinking together can shorten the process and it is not 
twice the effort? Beck indicates that through design 
standards and other complementary practices, you will 
not necessarily double effort. In fact, overall calendar 
time (i.e., duration) for developing a bit of code could 
shrink. He actually makes the strong assertion that 
“pair programming is more productive than dividing 
the work between two programmers and then 
integrating the results.” (p.82) In this statement he is 
strongly arguing against the primary criticism of pair 
programming – that it simply takes too much effort. 
His argument was clearly that it does not. Quite the 
opposite, he believed pair programming is quicker.

So Beck made very specific arguments about pair 
programming. It will improve software quality, 
improve developer learning, and will not adversely 
affect duration (indeed, it may even improve 
productivity).

3. Testing Beck’s Argument

Meta-analysis is a statistical tool used to 
systematically integrate results of empirical research 
that are independent from each other but that address 
similar research questions [22, 27, and 31]. We used 
meta-analysis to analyze the results of previous studies 
on pair programming. Below we briefly outline the 
advantages of meta-analysis for addressing Beck’s 
assertions about pair programming.

First, meta-analysis, unlike traditional narrative 
reviews, allows a systematic and mathematical 
combination of mean differences across two (or more) 
variables [31]. In this paper, we computed the mean 
differences between pair and solo programming for 
three different performance variables: quality, learning, 
and duration. Second, meta-analysis is useful for 
detecting publication bias through the estimation of 
sampling error. Bias in publication can be detected 
through a scatterplot of the mean effect size by 
sampling error (larger studies possess smaller standard 
errors), called a funnel plot. If effect sizes in the funnel 
plot are not symmetrical and do not resemble an 
inverted funnel, then publication bias may exist [45]. 
Out of the three existing pair programming meta- 
analyses [19, 24, and 47] only one accounts for 
publication bias [24]. However, we are more 
conservative than [24] in how we address bias in 
publications. Instead of using the trim and fill method 
[18] we follow the recommendations of Lipsey and 
Wilson and others [31, 33, and 52] and perform a 
sensitivity analysis (i.e., we remove biased studies 
from our sample instead of imputing new values for 
them). Borenstein and associates [6] also favor 
sensitivity analysis over the trim and fill method as 
they argue that it would be hard for researchers to 
interpret the possible fluctuations in the weighted mean 
effect size.

In sum, a meta-analysis adds rigor and validity to 
our findings in a way that narrative reviews of the 
literature cannot match. Further, our approach involves 
a stricter test of existing findings than other previously 
published meta-analyses on the impact of pair 
programming.

3.1. Data Source

Following the recommendations of [57 and 27], we 
used a keyword to search for the literature across 
different electronic databases (e.g., Science Direct, 
Business Source Complete, PsycINFO) through May 
2015. We included conference proceedings, 
dissertations, and theses in our search mechanism as a 
means to address bias toward higher effects sizes 
typically associated with published journal articles 
[45]. To ensure the capture of all relevant articles, we 
also used citations in the articles we “pulled” to 
identify additional pair programming articles. Our first 
search yielded 244 articles. In a pre-screen of the 
stakes we deleted articles that included the keyword 
pair programming in the text but were not actually 
about pair programming [e.g., 28]. We also removed 
studies that were not written in English [e.g., 20]. This
resulted in the deletion of 16 articles, resulting in a final sample of 228 studies. For all of these identified articles we developed a rigorous set of inclusion criteria to evaluate their applicability and usefulness for our meta-analysis.

3.2. Inclusion Criteria

Five inclusion criteria were used to assess articles. First, studies had to define pair programming as a practice when two developers work together on the same task using one computer screen [61]2. Twenty-one studies did not meet this inclusion criterion.

Second, articles had to compare pair programming against solo programming. Studies that only focused on pair programming [e.g., 15] or did something else were dropped. Sixty-six studies did not meet this inclusion criterion.

Third, the article had to provide an independent dataset. This means that different articles containing the same dataset were eliminated to avoid bias through multiple counts [e.g., 12]. However, one article can contribute more than one set of mean differences if independent datasets were used [e.g., 62 contributed three datasets]. Thirteen studies did not meet this inclusion criterion.

Fourth, studies had to capture at least one of the following performance variables: quality, learning, and duration3. Articles that measured something else (e.g., self-efficacy) were excluded. Twelve studies did not meet this inclusion criterion.

Finally, articles had to provide means and standard deviations for both pair and solo programming performances for at least one of our dependent variables. Articles that reviewed the pair programming literature or were conceptual in nature were dropped [e.g., 9]. For empirical articles, we searched for the descriptive statistics table. If means and standard deviations were not presented in the article, we looked for other information (e.g., t-test and chi-square values). If these other analyses were not presented in the article, we excluded them from further examination. A total of 94 articles were excluded based on this criterion.

This resulted in a total of 22 papers, or 24 individual datasets, for the meta-analysis. Of these studies, 12 are journal articles and 10 are conference papers. Twenty-two studies is a large sample size compared to other meta-analyses on pair programming [e.g., [19] contained 16 empirical studies, [24] included 18 studies, and [47] analyzed 10 studies].

3.3. Coding Procedures

In addition to collecting basic information about every study (e.g., author, year, independent and dependent variables), we also coded relevant statistics (e.g., means, standard deviations, and sample sizes), measures for the three dependent variables (e.g., final exam score for learning), and any possible moderators (e.g., task complexity).

3.4. Meta-Analytic Approach

We used Lipsey and Wilson’s approach to meta-analysis [31]. Their technique follows that of Hunter and Schmidt in that it is also based on a random-effects model (for a discussion on this approach see [22 and 27]). In this study, the random-effects approach is more appropriate than a fixed-effects model because we accumulated data from a series of studies that were performed by other researchers. It is unlikely that all these studies were functionally equivalent. That is, the study participants or the manipulation in the studies we collected were different in many ways and these discrepancies probably had an impact on their results, and therefore we should not assume a common effect size for them [6].

We used SPSS to analyze the data using Lipsey and Wilson’s MeanES macros. We also created funnel plots to examine whether publication bias existed and used a calculator to compute failsafe N values. These are useful for examining the number of studies needed to nullify the mean effect sizes.

4. Results

The results of our meta-analysis are shown in Table 1. Our findings suggest that, on average, pairs produce higher quality programs than individuals ($ES = 0.31$, $weighed SD = 0.539$, $z-value = 1.93$, $p-value = 0.054$). About 182 additional negative studies (studies in which the mean differences between pairs and individuals are zero) would be necessary to reverse this positive effect, indicating that the effect is strong. We also find that pair programming is superior to solo programming when it comes to learning ($ES = 0.24$, $weighed SD = 0.259$, $z-value = 2.71$, $p-value = 0.06$). Finally, our results indicate that although pairs seem to write programs faster than individuals (negative coefficient), such effect is not statistically significant ($ES = -0.65$, $weighed SD = 1.00$, $z-value = -1.54$, $p-value = 0.12$).

---

2 See the Appendix for a list of all the studies included in our meta-analysis.

3 We excluded effort because it is different than duration. In most studies, for example, pair effort is measured as twice the duration of solo effort. In addition, we only found 2 eligible studies for effort [1 and 30], both of which reported positive results (i.e., pairs exert more effort than individuals).
Table 1. Meta-analysis results

Funnel plots for all dependent variables show publication bias evident in some studies (see Figures 1 through 3). We conducted a sensitivity analysis for each variable to address this issue.

Correcting for publication bias resulted in the removal of 10 results from our sample, 5 for quality, 1 for learning, and 4 for duration. This correction increases our confidence that pairs produce higher quality programs than individuals ($ES = 0.33$, weighed $SD = 0.32$, $z$-value = 2.61, $p$-value = 0.009). We reach the same conclusion for learning even though its effect size actually shrinks after the sensitivity analysis ($ES = 0.19$, weighed $SD = 0.215$, $z$-value = 2.37, $p$-value = 0.01). Interestingly, the corrected results for duration now show a less negative, yet statistically significant, effect size ($ES = -0.55$, weighed $SD = 0.126$, $z$-value = -2.14, $p$-value = 0.031). This means that, on average, pairs program faster than individuals.

Table 2. Meta-analysis results (Corrected for publication bias)

Results after bias correction are summarized in Table 2 and illustrated in Figures 4 through 6.
We also calculated a $Q$-test to examine whether we had heterogeneity in the mean effect sizes [26]. The $Q$-test is distributed as a chi-square with $k-1$ degrees of freedom, where $k$ is the number of effect sizes [31]. Heterogeneity will exist if the $Q$-test value is larger than the critical value in the chi-square distribution.

Our results show significant heterogeneity across quality and learning studies. The highest heterogeneity was found in quality ($Q = 21.04, p$-value = 0.007). The $Q$ value for learning was 15.65 with a $p$-value equal to 0.04. However, the $Q$-test only informs us whether heterogeneity exists, with no further details as to the extent in which it exists. That is why several researchers also report the $I^2$-squared index, which represents the percentage of total variability among effect sizes that is due to actual heterogeneity rather than just sampling error [26]. The lower the value of the index the lower the percentage of heterogeneity and the higher the sampling error percentage. [26] suggest that if $I^2$-squared equals zero, then all variability is due to sampling error, and not actual heterogeneity. In addition, [25] suggest that $I^2$-squared values of .25 (25%), .50 (50%), and .75 (75%) are classified as low, medium, and high heterogeneity, respectively.

The $I^2$-squared values for both quality and learning studies are medium ($I^2$-squared quality = 0.62 (62%); $I^2$-squared learning = 0.42 (42%)). This suggests that the
significant heterogeneity seen in the $Q$-test is not due to sampling error but perhaps some moderators.

5. Discussion and Conclusion

In 2000, Kent Beck suggested that pair programming will improve software quality, increase learning outcomes, and reduce duration of programming tasks. According to our meta-analysis on the subsequent research into this topic, he was correct. However, our further testing shows that this is probably not the end of the story—there are likely some moderators in each of these relationships. This means that the relationship between pair programming and organizational outcomes is a bit trickier—a point consistent with recent theorizing [48] suggesting that “pair programming is more effective than solo programming in improving quality when pairs are similarly skilled” [48, p.5].

5.1. What We Know

Our analysis shows that the data supports Beck’s argument about the effect of pair programming on quality, learning, and duration. These findings echo existing meta-analyses on pair programming [19, 24, and 47], but differ in that we are the first meta-analysis to report unbiased results for learning ([47] include learning but do not test for publication bias). In addition, our findings for quality are more robust than those presented by [24] since we did not interpret fluctuations in the mean effect sizes but instead took on a more conservative approach to publication bias by performing sensitivity analysis. The important part of this approach is that we managed to do it without losing statistical power. This is because we examined a larger number of studies than any previously conducted meta-analysis on pair programming.

Our findings have the further benefit of being consistent with research that did not make it into our sample because they did not meet our criteria (such as qualitative case-based analyses). A variety of studies not included in our sample found that pair programming improves quality relative to solo programming [16, 49, 14, 38, and 59]; that pair programming leads to better learning outcomes than solo programming [58, 49, 46, 53, 54, 55, and 47] and that pair programming can reduce total development time [32 and 56].

However, it is important to note that research is not know about the effects of pair programming.

5.2. What We Don’t Know

The medium $I$-squared values for both quality and learning suggest that moderating variables may explain some of the heterogeneity in our results. Task complexity and expertise are the two most popular moderators in the pair programming literature [24 and 48]. Pair programming is expected to improve quality for complex tasks, but not necessarily simple tasks. Similarly, more learning is expected to take place in complex tasks rather than simple ones. Further, expertise also matters. Experienced programmers will outperform others whether paired or not in terms of both software quality and duration [5]. Also, learning outcomes are thought to be maximized with at least one highly experienced developer. Recent thinking actually points to the possibility for two-way interactions between experience and complexity [48]. Yet, we still know very little about contingency effects on quality, learning, and duration. Researchers have simply not theorized much about interaction effects but have instead focused on the main effects [48]. Interaction effects are rarely tested in pair programming experiments. A meta-analysis on moderating factors still appears to be a long way off—we need empirical tests to start concentrating on moderators.

The variety of definitions and measures for the different dependent variables may also explain the heterogeneity in our results. Quality, for example, is measured in a couple of different ways. The most common single measure is defect density [50 and 51] which accounts for various sorts of errors per body of code. Broader efforts are also common, such as perceptual measure (e.g., Martin’s metrics) [36]. These
are conceivably different indicators for the same idea of quality, but quality can also mean many different things, and we do not know if differential definitions of quality impact our knowledge about the outcomes of pair programming.

Similarly, we may have issues in measuring duration and related variables. Duration, effort, and productivity are all related. Duration is the total calendar or wall time required to complete a task [7], productivity is the task per unit of time [51], and effort is the person hours required to complete a task [1 and 30]. These three variables are measured similarly in extant studies – one might argue that studies are measuring close to the same thing, given the nature of controlled lab experiments. For a set task, duration is the inverse of productivity, and half of effort for a pair programming task (duration is equal to effort for a solo programming task in a typical lab experiment).

We conducted a meta-analysis of the three variables that core to Beck’s introduction of the topic. These are three important variables to software development organizations and most existing studies address them. Yet, there are a number of other important factors that have been discussed in the literature and deserve attention such as satisfaction, enjoyment, and confidence [54, 44, 35, 29, and 17]. Are pairs more satisfied than solo programmers? Are they more confident? These are important variables that have not yet been thoroughly explored in the pair programming literature.

In short, while we do know some things (pair programming generally increases quality and learning, and reduces task duration), we still have much to learn. The next era of research into pair programming should attend directly to contingency variables and how these can affect the different outcomes of pairs (versus solo) in both the short- and long-term.

6. References


7. Appendix

<table>
<thead>
<tr>
<th>Study</th>
<th>DV(s)</th>
<th>N(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Akour et al. 2013</td>
<td>Learning</td>
<td>43</td>
</tr>
<tr>
<td>Arisholm et al. 2007</td>
<td>Quality</td>
<td>197</td>
</tr>
<tr>
<td>Balljepally et al. 2009</td>
<td>Quality</td>
<td>60</td>
</tr>
<tr>
<td>Bravo et al. 2013</td>
<td>Quality and Duration</td>
<td>51</td>
</tr>
<tr>
<td>Braught et al. 2011</td>
<td>Learning</td>
<td>176</td>
</tr>
<tr>
<td>Canfora et al. 2005</td>
<td>Duration</td>
<td>36</td>
</tr>
<tr>
<td>Canfora et al. 2007</td>
<td>Quality and Duration</td>
<td>18</td>
</tr>
<tr>
<td>Lai and Xin 2011</td>
<td>Learning</td>
<td>59</td>
</tr>
<tr>
<td>Lemos et al. 2012</td>
<td>Quality</td>
<td>46</td>
</tr>
<tr>
<td>Madeyski 2006</td>
<td>Quality</td>
<td>90</td>
</tr>
<tr>
<td>Madeyski 2007</td>
<td>Quality</td>
<td>98</td>
</tr>
<tr>
<td>Madeyski 2008</td>
<td>Quality</td>
<td>98</td>
</tr>
<tr>
<td>Maguire et al. 2014</td>
<td>Learning</td>
<td>220</td>
</tr>
<tr>
<td>McDowell et al. 2003</td>
<td>Quality and Learning</td>
<td>486</td>
</tr>
<tr>
<td>Mueller 2006</td>
<td>Quality</td>
<td>18</td>
</tr>
<tr>
<td>Nosek 1998</td>
<td>Quality and Duration</td>
<td>15</td>
</tr>
<tr>
<td>Owolabi et al. 2013</td>
<td>Learning</td>
<td>51</td>
</tr>
<tr>
<td>Sison 2008</td>
<td>Quality</td>
<td>24</td>
</tr>
<tr>
<td>Sison 2009</td>
<td>Quality and Duration</td>
<td>21</td>
</tr>
<tr>
<td>Williams et al. 2002</td>
<td>Learning</td>
<td>113</td>
</tr>
</tbody>
</table>

| Williams et al. 2003  | Learning | 158, 224, 165 |
| Xu and Rajlich 2006   | Duration | 12           |

Table 3. Studies included in the meta-analysis