Effectiveness of Pair and Solo Programming Methods: A Survey and an Analytical Approach

Full papers

Wenying Sun
Washburn University
nan.sun@washburn.edu

Miguel I. Aguirre-Urreta
Texas Tech University
miguel.aguirre-urreta@ttu.edu

George M. Marakas
Florida International University
gmarakas@fiu.edu

Abstract

We conducted two studies to further our understanding of the dynamics of the programming methods. One is a survey study. We surveyed software professionals in the industry to collect their views on the effectiveness of pair programming versus solo programming. In our second study, we adopted the analytical approach to compare the three modes of programming: solo only, pair only, and a mixture of solo and pair. The second study involves three steps. First, we replicated the study conducted by Dawande and colleagues (Dawande et al. 2008). Second, we applied the parameters collected from our survey to the same model. Third, we extended the analytical model to further study the effect of project complexity and pair composition on the effectiveness of the three different programming modes. Due to space limitations, in this paper, we only report: a) the survey research and its findings; b) partial results from step 2 and step 3 of the second study.

Keywords

Pair programming, solo programming, effectiveness, practitioner survey, simulation

Introduction

Pair programming is a programming approach where two programmers work together on one programming task, with one as the driver controlling the keyboard, and the other as the navigator providing alternatives to the programming task (Arisholm et al. 2007; Beck and Andres 2004; Dybå et al. 2007). With the increased popularity of agile software development methods, pair programming appears to have become a fairly attractive alternative to the traditional practice of solo programming.

However, pair programming is characterized by having a rich body of literature with little consensus on some important issues (Sun 2011). Prior research studied the effectiveness of pair programming vs. solo programming on a variety of factors, e.g. effort, defect rate, knowledge transfer, cost, job satisfaction. Almost on all aspects, some studies stated pair programming was more effective than solo programming while others identified contradictory results. In addition, a thorough review of the literature suggests existing research primarily used experiments and case studies as means to collect and report data. There have been very few survey research or analytical approaches to the study of programming methods. Furthermore, mixed developer assignment (a mixture of solo programming and pair programming) is rarely investigated.

To extend our understanding of the dynamics of the programming methods, we conducted two studies. The first is a survey study. We surveyed software professionals in the industry to collect their views on the effectiveness of pair programming compared to solo programming. In our second study, we adopted an analytical approach to better understand the outcomes of three different modes of programming: solo

only, pair only, and a mixture of solo and pair. We took three steps in our second study. First, we replicated the study conducted by Dawande and colleagues (Dawande et al. 2008). The purpose of the replication is to ensure we understood the logic of the original analytical model and were able to produce the same results. Second, we applied the parameters collected from our survey to the same model. The purpose of this step is to identify the effect of different parameters on the model and consequently explain the differences in results. Third, we extended the analytical model to further study the effect of project complexity and pair composition on the effectiveness of the three different programming modes. Due to space limitations, in this paper, we only report: a) the survey research and its findings; b) partial results from step 2 and step 3 of the second study.

The rest of the paper is organized as follows. We first introduced the survey study, the respondents' profile, and its findings. We then describe the design and implementation of the analytical approach to the study of the three programming modes. Next, we report the simulation results on total programming effort and pair formation effort. Lastly, we discuss the findings.

Survey Results

We conducted a survey of software professionals in industry to collect their views on the effectiveness of pair programming as compared to solo programming. Nearly 1,500 email addresses were collected through professional membership associations and their conferences. An email was sent to request participation in this survey research. A reminder was sent two weeks later to those who did not participate. Data collection was conducted through an online survey tool and lasted for a month. We received 194 complete responses, out of which 131 respondents had pair programming experience. Even though professionals without pair programming experience could undoubtedly form legitimate perceptions of pair programming without direct experience, for later analysis, we decided to use responses from professionals with pair programming experience only. The results reported below came from those professionals.

Table 1 presents the demographic characteristics of the respondents. The profile suggests the responses came from well-educated and experienced practitioners, with 87.8 percent of the respondents having college or graduate degrees, and 77.8 percent having over 10 years of working experience in the information technology field. The pair programming experience reported by respondents ranges from less than a year to more than 10 years. On average, the respondents have four years of pair programming experience.

Respondents					
	N	%		N	%
Pair programming experience			**Experience in IS/IT field**		
<1 year	28	21.3%	< 2 years	1	0.7%
1-2 years	34	25.9%	2-5 years	4	3.0%
3-4 years	28	21.3%	6-10 years	24	18.3%
5-6 years	22	16.7%	11-20 years	62	47.3%
7-8 years	7	5.3%	20+ years	40	30.5%
9-10 years	4	3.0%			
>10 years	8	6.1%			
Age			**Highest education**		
<30	13	9.9%	High school or equivalent	1	0.8%
30-39	52	39.7%	Some college	15	11.5%
40-49	45	34.4%	College degree	50	38.2%
50-59	18	13.7%	Graduate degree	65	49.6%
60-69	2	1.5%	**Location**		
>69	1	0.8%	North America	109	83.2%
Gender			Europe	9	6.9%
Male	119	90.8%	Asia	10	7.6%
Female	12	9.2%	Australia/NZ	3	2.3%
Industry			**Current Position**		
e-Commerce	8	6.1%	Developer	14	10.9%
Financial	16	12.2%	Project Manager	29	22.1%

Government	5	3.8%	Team Leader	9	6.9%
IT Consulting	24	18.3%	VP/Director of Dev	21	16.0%
Manufacturing	8	6.1%	Development Manager	8	6.1%
Retail	1	0.8%	Consultant/Trainer	14	10.7%
Technology	48	36.6%	Architect	12	9.2%
Other	16	12.2%	QA/Tester	13	9.9%
Not specified	5	3.8%	Product Manager	3	2.3%
			IT Staff	1	0.8%
			CIO/CTO/CEO/President	1	0.8%
			Other	6	4.6%

Table 1. Profile of Survey Respondents

On the survey, we asked the respondents to estimate the percentage difference between solo programming and pair programming on programming effort, defect rate, and knowledge transfer in scenarios of projects at different levels of complexity (low, medium, high) and pairs comprised of different expertise (junior-junior pair, junior-senior pair, senior-senior pair; pair where none has prior pair programming experience, pair where one has prior pair programming experience, and pair where both have prior pair programming experience). Programming effort is the total amount of programmer-hours devoted to completing a project. Defect rate is the number of defects per thousand lines of code (a commonly used metric in software engineering). Finally, knowledge transfer captures the amount of learning that results from working in pairs. Each of these estimates were obtained with a single-item question that was original to this research effort. The measurement scales are from 81-99% decrease (-5), to no difference (0), to >100% increase (6) (numbers within parentheses indicate how each response was subsequently coded for later analysis). Due to space limitations, we are not attaching the survey instrument. A copy is available upon request.

The respondents generally believe pair programming will increase knowledge transfer and reduce defect rates. Furthermore, they believe this effect will be stronger when the project complexity is high, and when the pairs are comprised of two seniors where both have prior pair programming experience. For knowledge transfer, junior-senior composition works leads to the most learning. In terms of effort, survey respondents do not believe pair programming will reduce effort in all scenarios; rather, it will only reduce effort when there is a medium or high complexity project, and the pairs are comprised of junior and senior, or two seniors, and both have prior pair programming experience. Table 2 is a summary of the survey results.

Outcome	**Effort**	**Defect**	**Knowledge**
Complexity			
Low	0.85	-1.6	1.26
Medium	-0.24	-2.17	1.7
High	-1	-2.54	2.07
Pair Expertise			
Junior-Junior	1.04	-1.1	1.23
Junior-Senior	-0.4	-2.17	2.1
Senior-Senior	-0.88	-2.75	1.74
Prior PP Experience			
Neither-has	1.35	-0.93	1.18
One-has	0.02	-1.88	1.83
Both-have	-1.2	-2.63	1.9
*Measurement scales: No difference (0) Increase: 1-20% (1), 21-40% (2), 41-60% (3), 61-80% (4), 81-100% (5), > 100% (6) Decrease: 1-20% (-1), 21-40% (-2), 41-60% (-3), 61-80% (-4), 81-99% (-5)			

Table 2. Survey Results

Simulation Design and Implementation

The simulation reported here was custom coded in the R Statistical Environment. Many of the design decisions discussed next followed directly from the work of Dawande and colleagues (Dawande et al. 2008) in order to make our results more comparable. Where we deviated from their simulation design, this is explicitly noted and discussed in more detail.

The number of modules in the simulated system was either 40 or 70, representing small and large systems, respectively. Each of these modules was assigned to one of 6 different groups for purposes of between-module integration, where the number of modules in each group was similar. For example, in the case of 40 modules, there were four groups of seven modules each, and the remaining two groups contained six modules. Each of the modules was assigned to a complexity condition, being complex, average, or simple, which captures the base amount of development effort required. These were 1, 0.5 and 0.25 person-weeks, respectively. The total number of available developers was fixed at 10 in all simulation runs. Developers were generated by specifying the proportion of them that were junior developers (either 0.30 or 0.70), the proportion of them who had programmed in pairs before (either 0.30 or 0.70) and the proportion of those who had programmed together before (either 0.30 or 0.70), for which there would be no pair formation overhead. Assignment of developers to modules followed one of three conditions: all developers working alone (solo), all developers working in pairs (pairs) and mixed assignment (mixed); for the latter, the code ensured that in no case were all developers assigned on their own or in pairs. All developer assignment was done randomly.

For the case of a solo developer, the speed with which a module was completed was a function of developer expertise, which was 1 for senior developers and 2/3 for junior developers. When working in pairs, in addition to developer expertise, speed of completion also included multipliers based on our survey data, capturing the effects pair programming with regards to module complexity, pair composition, and prior pair programming experience, as previously discussed. Finally, we distinguish clock time (the time taken to complete development of a module) from total time (the total number of person-weeks required to complete the development). For the case of a solo programmer, they are identical. For the case of a pair, total time is clock time times two, which captures the fact that two developers were involved in the process.

The simulation code ensured that there were no standalone modules. The number of integration links between modules was either 0.05 or 0.10 of the total number of possible between-module links. In order to create simulated systems where within-group modules were more closely related than between-group ones, the number of integration links within a group was four times as large as the number of integration links between any two groups. The amount of work required to integrate any two modules was set at ten percent of the original amount of work needed to develop them in the first place. Calculation of the speed at which integration was completed was similar to the case of original module development just discussed, with the following differences. First, developers were randomly assigned to module integration based on the assignment condition (e.g., all solo, all pairs, mixed). Second, the possibility of overlap between the developers involved in each of the original modules and those assigned to integrate them was considered. For each overlap the speed of completion was increased by either 0.02 (low knowledge-sharing) or 0.30 (high knowledge-sharing). Further, these coefficients were multiplied by either 0.80 (when the shared developer was a junior developer) or 1.2 (when the shared developer was a senior developer).

Both original module development and integration work were considered subject to defects, which required a second round of work. The amount of work required was a function of the expertise of the original developers in each case, on the assumption that senior and junior developers generate defects at their stated productivity rate. As well, for the case where the original work was done in pairs, data from our survey was used to calculate the work necessary to fix the defects, based on module complexity, pair composition, and prior pair programming experience. The base defect rate for all conditions was also obtained from survey responses. Developers were randomly assigned to fix defects in both original development and integration work, separately for each. The speed with which the work was completed was, as before, a function of developer expertise, pair considerations for the case where pairs were assigned, and the amount of developer overlap between the original developers and those assigned to correct defects.

Finally, we also considered the effects of pair formation. At the beginning of each simulation run, some developers were randomly created as having prior pair programming experience. Of those, some of the developers (in both cases in the proportions discussed before) were assigned as having actually worked together before. When those developers were assigned to work in pairs in one or more of the four rounds of work previously discussed (original module development, defect fixing for original development, integration work, and defect fixing for integration work), they did not incur any pair formation efforts. For any other cases, the first time a pair of developers who had not worked together before was assigned to work together, pair formation overhead was incurred (the pair experience matrix was subsequently updated in order to avoid incurring overhead more than once). The amount of time taken to form a pair was either 0.10 or 0.50 person weeks, following prior work (Dawande et al. 2008). Further, these times were multiplied by 1.2 (when the new pair included two junior developers), 1 (when the new pair had one junior and one senior developer) and 0.80 (when both developers in the new pair were senior).

The complete simulation designed examined 384 different conditions: number of modules (2) x link density (2) x knowledge-sharing (2) x pair formation overhead (2) x proportion of junior developers (2) x proportion of developers with prior pair programming experience (2) x proportion of those who had worked together before (2) x developer assignment (3 for all solo, all pairs, and mixed). For each one of these combinations we replicated the results 1000 times, to account for variability in developers and their assignment to different modules.

Simulation Results

The data obtained from the simulation were analyzed by means of a mixed multi-factor ANOVA. In the simulation design, each of the design conditions, with the exception of developer assignment mode (that is, all pairs, all solos, or mixed), were replicated 1000 times. Each of these replications was then processed using all three of the developer assignment modes. As a result, each of the 128,000 replications generated three sets of results: when all developers in all stages were working on their own, were working in pairs, or when mixed assignment was possible. In our mixed ANOVA analysis, the developer assignment mode was thus the within-subject factor, and the remaining simulation conditions were treated as between-subject factors. For each of the different developer assignment modes, we focused on two main results: total project effort (the number of person-weeks that would be needed to complete one run of the simulated project) and pair formation time (the total amount of time taken by new pairs to start working together; for the case of all developers working on their own this was always zero). Given the large number of possible combinations of between- and within-subject factors, we focus here only on those results that are both significant and of practical interest, given by an effect size that is medium or larger. Effect size is given by partial eta-squared, which is the proportion of the sum of the effect and error variance that is attributable to the effect and is expressed with values in the 0 to 1 range. Interpretation guidelines are given as 0.02, 0.13 and 0.26 for small, medium, and large effects, respectively. Particular emphasis is placed on results that involve developer assignment mode, as that is the main subject of this research.

Total Project Effort

The first measure of interest in this research is Total Project Effort, which captures the total number of person-weeks that were consumed by the project through its four stages of module development, module defect fixing, between-module integration, and integration defect fixing. For the case of developers working alone, these figures are identical to those discussed in the previous section. In the case of pairs, the total number of person-weeks involves doubling the time taken by a pair to complete an assignment. Therefore, this measure is related to the allocation of resources in terms of person-weeks needed to complete the project. Table 3 presents the results of employing the different developer assignment modes and their interactions with various design conditions.

Simulation Condition	Developer Assignment Mode		
	All Solo	Mixed	All Pairs
Mean across all conditions (.955)	60.29	56.39	52.63
Number of modules (.725)			
Small (40 modules)	39.41	36.87	34.44
Large (70 modules)	81.18	75.90	70.82

Link density (.211)			
Low (5% of possible links)	54.05	50.57	47.24
High (10% of possible links)	66.54	62.20	58.02
Proportion junior developers (.831)			
Low (30% of developers)	56.61	50.80	45.25
High (70% of developers)	63.98	61.97	60.01
Past pair experience (.890)			
Low (30% of developers)	60.24	58.76	57.30
High (70% of developers)	60.35	54.01	47.96
Number of modules x Prop. junior developers (.319)			
Small x Low	37.28	33.44	29.77
Small x High	41.54	40.31	39.12
Large x Low	75.93	68.16	60.74
Large x High	86.43	83.64	80.89
Number of modules x Past pair experience (.527)			
Small x Low	39.81	38.80	37.82
Small x High	39.01	34.94	31.07
Large x Low	80.66	78.72	76.79
Large x High	81.70	73.08	64.85
Prop. junior developers x Past pair experience (.131)			
Low x Low	56.53	52.84	49.27
Low x High	56.68	48.76	41.24
High x Low	63.94	64.69	65.34
High x High	64.03	59.26	54.68
Note: Results are average person-weeks to complete a simulated project under the specified conditions. Partial eta-squared effect size given in parenthesis next to each condition.			

Table 3. Results for Total Project Effort

First, there is a strong main effect of developer assignment mode on the number of person-weeks necessary: developers working in pairs result in an overall decrease in the amount of resources needed to complete a project.

Second, there are also some first order interactions with project size (number of modules), link density between those modules, and the proportion of developers who are juniors (e.g., less experienced and therefore more error prone and less productive, as discussed in the simulation design). There are slight improvements in favor of pair programming when the projects are larger and the modules more interconnected, and a marked improvement when the proportion of junior developers is relatively low.

Third, there are three second-order interactions of interest, which involve project size and the proportion of junior developers, project size and the proportion of developers that have prior pair programming experience, and the interaction between these two factors (proportion of junior developers and the proportion of developers that have prior pair programming experience). The first of these three interactions shows that pair programming performs better when the proportion of junior developers is relatively low, for both project sizes. The second one indicates that pair programming also performs better when compared to solo when the proportion of developers with past pair programming experience is relatively high (note that in all these cases pair programming also performs better than solo programming for all conditions, but not markedly so). Finally, the last interaction indicates that pair programming is particularly effective when the proportion of junior developers in the development team is relatively low and the proportion of developers with prior pair programming experience is relatively high. Notably, pair programming performs worse than solo programming – that is, requires more person-weeks allocated to complete a project under these conditions – when the converse occurs.

Pair Formation Time

The second major outcome of interest in our simulation was Pair Formation Time, which captures the number of person-weeks that would be consumed by developers who have been assigned to work together in a pair but which have not done so in the past. This effect, also known as 'pair jelling', has been observed in the literature, but can exhibit a fair amount of variability, at least from published results (Williams et al. 2000). In our simulation design, we distinguished between having prior pair programming experience – that is, having programmed in pairs before, and thus being more familiar with what pair programming entails – from having (or not) programmed with a particular other developer before, which is what we consider here. As a result, the simulation condition "Past pair experience" captures whether developers have programmed in pairs before, either 30 or 70% of them, for the Low and High conditions respectively. On the other hand, the simulation condition "Proportion of existing pairs" represents the extent to which developers who do have prior pair programming experience have worked in a pair with another developer in the current team before, also in Low and High conditions as before. Results shown in Table 4 represent the average amount of person-weeks needed for new pairs to begin working effectively together, under a variety of simulation conditions and only for those where the effect size (as measured by the partial eta-squared statistic) can be considered medium or larger based on commonly accepted interpretation guidelines.

Simulation Condition	Developer Assignment Mode	
	Mixed	**All Pairs**
Mean across all conditions (.998)	10.84	11.45
Number of modules (.549)		
Small (40 modules)	10.23	11.37
Large (70 modules)	11.45	11.53
Link density (.224)		
Low (5% of possible links)	10.54	11.39
High (10% of possible links)	11.15	11.51
Pair formation (.995)		
Low (0.10 person-weeks)	3.61	3.82
High (0.50 person-weeks)	18.07	19.08
Proportion junior developers (.768)		
Low (30% of developers)	9.92	10.47
High (70% of developers)	11.76	12.42
Past pair experience (.878)		
Low (30% of developers)	12.20	12.88
High (70% of developers)	9.48	10.01
Proportion existing pairs (.697)		
Low (30% of developers)	11.61	12.26
High (70% of developers)	10.07	10.63
Number of modules x Link density (.148)		
Small x Low	9.68	11.25
Small x High	10.77	11.48
Large x Low	11.39	11.53
Large x High	11.52	11.53
Number of modules x Pair formation (.351)		
Small x Low	3.41	3.79
Small x High	17.05	18.94
Large x Low	3.82	3.84
Large x High	19.09	19.21
Pair formation x Proportion junior developers (.596)		

Low x Low	3.31	3.49
Low x High	3.92	4.14
High x Low	16.53	17.45
High x High	19.61	20.70
Pair formation x Past pair experience (.762)		
Low x Low	4.07	4.29
Low x High	3.16	3.34
High x Low	20.34	21.47
High x High	15.80	16.68
Pair formation x Proportion existing pairs (.506)		
Low x Low	3.87	4.09
Low x High	3.36	3.54
High x Low	19.35	20.43
High x High	16.78	17.72
Past pair experience x Prop. existing pairs (.589)		
Low x Low	12.37	13.06
Low x High	12.04	12.71
High x Low	10.86	11.46
High x High	8.10	8.55
Pair formation x Past exp. x Existing pairs (.389)		
Low x Low x Low	4.12	4.35
Low x Low x High	4.01	4.24
Low x High x Low	3.62	3.82
Low x High x High	2.70	2.85
High x Low x Low	20.61	21.76
High x Low x High	20.07	21.18
High x High x Low	18.10	19.11
High x High x High	13.50	14.26

Note: Results are average person-weeks to form pairs where assigned developers had not worked together before. Partial eta-squared effect size given in parenthesis next to each condition. The 'all solo programmers' condition omitted since pair formation time was not applicable to that condition.

Table 4. Results for Pair Formation Time

First, the results presented in Table 4 do not include the 'all solo programmers' condition since, for all simulation conditions (including those not reported here) the average time over all replications is exactly zero for the case of all developers working on their own. This should be the case, since solo developers do not, by definition, need to incur time learning to work with other developers, and validates that both the pair assignment mechanism as well as the calculation of time in the simulation works as intended.

Second, the time in the mixed assignment condition – where some developers are assigned to work in pairs and some on their own – is always less than the time taken to form pairs in the case where all developers must work in pairs. This is also as expected, since the simulation design enforced that, under the mixed assignment condition, there would be no cases of all developers working on their own or all developers working in pairs. This further validates that the simulation run as intended. The remainder of the results will be discussed in terms of their effect on the case of all developers working in pairs.

Third, there are a few important first-order interactions, in that the is greater when there are more modules in the project of the density of the links between them is greater; both of these correspond to more opportunities for developers to work in pairs and, everything else being equal, resulting in a greater number of possibilities for new pairs to be formed. The increase in time due to the greater presence of junior developers is a result of our simulation design, which considers that junior developers would take

longer than their senior counterparts to get accustomed to working with another developer, given their overall limited experience in development in general. There is also a marked effect due to the particular value of the simulation condition that specifies the base amount of time needed for a new pair to form (e.g., 0.10 or 0.50 person-weeks), which is then adjusted by other factors such as the overall experience of the developers involved in the particular pair, as just discussed. Also as expected the proportion of programmers who have worked in pairs before and the proportion of programmers who have worked with other programmers in their own team before has an effect in the results.

Finally, there are a number of second-order and one third-order interaction which capture many of the effects discussed before (opportunities to work in pairs, existing pair programming experience, developer expertise, etc.) together with the base cost of forming a new pair. More interestingly, our simulation seems to capture well the diversity of times that has been observed in the literature. For example, our results go from a low of 2.85 person-weeks as the average total time to form all pairs in a project when the base cost is low, past pair programming experience is high, and the proportion of developers who have worked together in pairs is high, to a high of 21.76 person-weeks when the opposite conditions are observed. Our results thus indicate that the time incurred in forming new pairs can vary substantially depending on the particular characteristics of the project and developer team, and should thus be an important consideration when choosing among alternative assignment alternatives. At the same time, pair formation is essentially a one-time cost and therefore, over time as more and more developers in a development team have worked together in a variety of projects, unlikely to be a major determinant of the project budget or schedule.

Discussion and Conclusions

In this research we sought to contrast and compare two alternative approaches to software programming, solo development versus development conducted in pairs, which is an issue that has garnered some attention since pair programming was introduced as an alternative to traditional development practices. We do this by means of a simulation that combines the design originally developed by Dawande and colleagues (Dawande et al. 2008) with results from our own survey of software practitioners, which provide realistic values for many of the parameters in the simulation. Given that these were obtained from professionals with both extensive industry experience as well as actual experience in pair programming, we believe this contributes to the external validity of our findings.

Due to space limitations we focus here solely on the major results of interest, Total Project Effort and Pair Formation Time, at the expense of others that were obtained from the simulation; for instance, we are not considering here the variability of our results, and focus solely on the average values obtained. These are further analyzed by means of a mixed-methods ANOVA, with also an emphasis on those results that pertain to developer assignment mode, our main topic of interest here. As our results indicate, pair programming should be considered a viable alternative to traditional solo development in the majority of cases, but more so under more particular conditions. For example, the difference between solo and pair programming, for the same group of developers, is magnified when projects are larger, developers have more expertise and they have a stronger background in pair programming itself. Solo development, on the other hand, performs relatively better (or at least not markedly different than pair programming) when the opposite conditions apply. The main result of interest is that differences between the two approaches appear to be notably dependent on the particulars of both the project under consideration and the developers that are available to complete it, which suggests that managers would do well to consider these specifics because deciding on which approach is more or less appropriate at any given time. We also examined here Pair Formation Time, which captures the time taken for a new pair – two developers that have not worked together before, to coalesce and begin functioning productively. Our results show, consistent with past research, that there is quite a fair amount of variability depending on various characteristics of the particular group of developers under consideration, which should also factor in the abovementioned decision. It should also be noted that this 'cost' incurred by new pairs is a one-time occurrence and that, over time and as pair programming becomes more prevalent within a given development team, its impact on the development approach decision decreases quickly. Given that this is one of the central decisions faced by managers of software development teams, we hope our ongoing research in this area may prove useful to practitioners in the future.

REFERENCES

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

Beck, K., and Andres, C. 2004. *Extreme Programming Explained: Embrace Change*. Addison-Wesley Professional.

Dawande, M., Johar, M., Kumar, S., and Mookerjee, V. S. 2008. "A Comparison of Pair Versus Solo Programming under Different Objectives: An Analytical Approach," *Information Systems Research* (19:1), pp. 71-92.

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

Sun, W. 2011. "The True Cost of Pair Programming: Development of a Comprehensive Model and Test." PhD Dissertation, University of Kansas.

Williams, L., Jeffries, R., Kessler, R. R., and Cunningham, W. 2000. "Strengthening the Case for Pair Programming," *IEEE software* (17:4), pp. 19-25.