Information and Software Technology 51 (2009) 1110–1122

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

The effectiveness of pair programming: A meta-analysis

Jo E. Hannay [a,b,*], Tore Dybå [a,c], Erik Arisholm [a,b], Dag I.K. Sjøberg [a,b]

[a] Simula Research Laboratory, Department of Software Engineering, Pb. 134, NO-1325 Lysaker, Norway
[b] University of Oslo, Department of Informatics, Pb. 1080 Blindern, 0316 Oslo, Norway
[c] SINTEF Information and Communication Technology, NO-7465 Trondheim, Norway

ARTICLE INFO

Article history:
Received 20 October 2008
Received in revised form 9 February 2009
Accepted 9 February 2009
Available online 20 February 2009

Keywords:
Pair programming
Evidence-based software engineering
Systematic review
Meta-analysis
Fixed effects
Random effects

ABSTRACT

Several experiments on the effects of pair versus solo programming have been reported in the literature. We present a meta-analysis of these studies. The analysis shows a small significant positive overall effect of pair programming on quality, a medium significant positive overall effect on duration, and a medium significant negative overall effect on effort. However, between-study variance is significant, and there are signs of publication bias among published studies on pair programming. A more detailed examination of the evidence suggests that pair programming is faster than solo programming when programming task complexity is low and yields code solutions of higher quality when task complexity is high. The higher quality for complex tasks comes at a price of considerably greater effort, while the reduced completion time for the simpler tasks comes at a price of noticeably lower quality. We conclude that greater attention should be given to moderating factors on the effects of pair programming.

Contents

* Corresponding author. Address: Simula Research Laboratory, Department of Software Engineering, Pb. 134, NO-1325 Lysaker, Norway.
E-mail address: johannay@simula.no (J.E. Hannay).

0950-5849/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.infsof.2009.02.001

1. Introduction

Much of the current interest in pair programming is likely due to the popularity of extreme programming (XP), of which pair programming is one of the core practices [2]. Common-sense, but scientifically unwarranted claims as to both the benefits and the adversities of pair programming abound. Advocates of pair programming claim that it has many benefits over individual programming when applied to new code development or when used to maintain and enhance existing code. Stated benefits include higher-quality code in about half the time as individuals, happier programmers, improved teamwork, improved knowledge transfer, and enhanced learning [53]. There are also expectations with respect to the benefits and drawbacks of various kinds of pairing, e.g., that "expert–expert pairings seem to be especially accelerated" (*ibid*., p. 102) and that "novice–novice is the most problematic" (*ibid*., p. 120). Stephens and Rosenberg [49] claim that pair programming is clearly the least beneficial of XP practices and also that novice–novice pairing is obviously an undesirable combination, because it is akin to "the blind leading the blind".

With the considerable interest in pair programming by the software industry and academia, it is important to determine scientifically, whether, and if so when, pairing two programmers is beneficial in terms of important cost drivers such as duration, effort, and quality. The finding across decades of small group research is that groups usually fall short of reasonable expectations to improved performance [48,35,33]. An important question, therefore, is whether the claims regarding pair programming can be substantiated by empirical evidence and how pair programming relates to such group research. Several empirical studies have now been conducted that set out to examine the effects of pair programming in a systematic manner. This article provides a systematic review in terms of a meta-analysis of all published experiments on the effectiveness of pair programming and subsequently offers recommendations for evidence-based practice [18]. This review extends the intermediate analysis reported in [15], which briefly summarized pair programming experiments published up and until 2006. In the present full-scale analysis, we take into account between-study variance, subgroup differences and publication bias. We also take into account studies published in 2007.

Section 2 presents the methods for systematic review and statistical meta-analysis. Section 3 presents the results from the analysis, and Section 4 discusses implications for theory and practice. Section 5 concludes.

2. Method

Informed by the general procedures for performing systematic reviews [34] and the established methods of meta-analysis [39,45], we undertook the meta-analysis in distinct stages: identification of inclusion and exclusion criteria, search for relevant studies, identification and selection of studies, data extraction, and statistical analysis and synthesis, see details below.

The meta-analysis focused on combining quantitative effects on three central outcome constructs that were investigated in the included studies. We did not assess the quality of the included studies in terms of, e.g., the appropriateness of the chosen effect size measures [30], the appropriateness of randomization procedures [31], subject/task selection and validity issues [46], statistical power [17], the use of theory [25], the approach to realism [24], etc. Future meta-analyses might incorporate study quality, but at present, it is not clear how to aggregate this multifaceted concept into a single measure to be used in a meta-analysis.

2.1. Inclusion and exclusion criteria

Studies were eligible for inclusion in the meta-analysis if they presented quantitative data on the effectiveness of pair programming in which a comparison was made between pairs and individuals, possibly in a team context. The subjects could be either students or professional software developers. Included studies had to report one of the primary outcomes *Quality*, *Duration*, or *Effort*. We did not want to put any restrictions on the operationalization of these outcome constructs. Furthermore, the studies had to be reported in English.

2.2. Data sources and search strategy

The search strategy included electronic databases and hand searches of conference proceedings. We searched in the following electronic databases: ACM Digital Library, Compendex, IEEE Xplore, and ISI Web of Science. We did not perform separate searches in the SE-specific databases Kluwer Online, ScienceDirect, SpringerLink, and Wiley Inter Science Journal Finder, because previous experience with systematic search strategies has shown that articles retrieved from these databases are also returned by either ISI Web of Science or Compendex [16]. In addition to the electronic databases, we hand-searched all volumes of the following thematic conference proceedings: XP, XP/Agile Universe, Agile, and Agile Development Conference. We used the basic search string "'pair programming' OR 'collaborative programming'" to conduct the searches.

2.3. Study identification and selection

The identification and selection process consisted of three stages. At Stage 1, the second author applied the search string to the titles, abstracts, and keywords of the articles in the included electronic databases and conference proceedings. All retrieved articles were published, or accepted for publication, before or in 2007. Excluded from the search were editorials, prefaces, article summaries, interviews, news items, correspondence, discussions, comments, reader's letters, and summaries of tutorials, workshops, panels, and poster sessions. This search strategy resulted in a total of 236 unique citations.

At Stage 2, the first and second authors went through the titles and abstracts of all the studies resulting from stage 1 for relevance to the review. If it was unclear from the title, abstract, and keywords whether a study conformed to our inclusion criteria, it was included for a detailed review. This screening process resulted in 57 citations that were passed on to the next stage.

At Stage 3, the full text of all 57 citations from Stage 2 were retrieved and reviewed in detail by the first and second authors. This resulted in 23 included articles according to the inclusion criteria. Five of these did not report enough information to compute standardized effect sizes and were excluded. Thus, 18 studies (all experiments) met the inclusion criteria and were included in the review (see Appendix).

2.4. Data extraction and checking

We collected data from the 18 articles, including type of treatment, type of system, type of tasks, duration of the study, number of groups, group assignment, type of subjects and their experience with pair programming, number of pairs, number of individuals, outcome variable, means, standard deviations, counts, percentages, and *p*-values. Every article included in the review was read in detail and the data was extracted and cross-checked by the first, second and third authors. Discrepancies were resolved by discussion among all four authors.

2.5. Statistical analysis

We used Comprehensive Meta-Analysis *v*2 to calculate effect size estimates for all the tests in the 18 articles.[1] In order to be comparable across studies, effect sizes must be standardized. In this meta-analysis, we used Hedges' g as the standardized measure of effect size. Like Cohen's d and Glass' Δ, Hedges' g is simply the difference between the outcome means of the treatment groups, but standardized with respect to the pooled standard deviation, s_p, and corrected for small sample bias [37]:

$$\text{Hedges' } g = \frac{\bar{x}_1 - \bar{x}_2}{s_p} \tag{1}$$

The pooled standard deviation is based on the standard deviations in both groups, s_1, s_2:

$$s_p = \sqrt{\frac{(n_1 - 1)s_1^2 + (n_2 - 1)s_2^2}{(n_1 - 1) + (n_2 - 1)}} \tag{2}$$

Hedges' g, Cohen's d, and Glass' Δ have the same properties in large samples (i.e., they are equivalent in the limit $(n_1 + n_2) \to \infty$), but Hedges' g has the best properties for small samples when multiplied by a correction factor that adjusts for small sample bias [26]:

$$\text{Correction factor for Hedges' } g = 1 - \frac{3}{4(N-2) - 1}, \tag{3}$$

where N = total sample size.

An effect size of .5 thus indicates that the mean of the treatment group is half a standard deviation larger than the mean of the control group. Effect sizes larger than 1.00 can be assumed to be *large*, effect sizes of .38–1.00 *medium*, and effect sizes of 0–.37 *small* [30].

2.6. Synthesis

We conducted separate meta-analyses for the three outcome constructs *Quality*, *Duration*, and *Effort*. Some studies applied several tests on the same outcome construct. In theses cases, we used the mean of the effect sizes over these tests to give only one effect size per outcome per study. Comprehensive Meta-Analysis calculates these mean effect sizes as an option.

Because we expected considerable heterogeneity, we decided to calculate the resulting meta-analytic effect sizes both under the assumption of the random-effects model and under the assumption of the fixed-effects model. These models lead to different significance test and confidence intervals for meta-analytic results [29].

The fixed-effects model assumes an unknown and fixed population effect size that is estimated by the studies in the meta-analysis. All the studies in the meta-analysis are seen as drawn from the same population, and variances in effect sizes between individual studies are viewed as due to subject variability [27].

The random-effects model, on the other hand, assumes an unknown and stochastic population effect-size distribution. That is, the true effect size of pair programming varies around a mean μ. This caters for the view that the effect of pair programming depends on situational variables and other factors (both known and unknown) that are not taken into consideration in the analysis. Variance in effect sizes are then seen as due to subject variability, and also to inter-study variability, since each study is seen as approximating a different part of the true effect size distribution [27,4].

Both models relate to an unknown population parameter. The approaches are hence referred to as *unconditional* [37]. The choice of which model to use is made prior to the analysis based on theory, past empirical findings, and on insights as to what the included studies describe.

However, one may also use the analysis techniques associated with the two models merely to characterize the studies relatively to each other without any reference to a population effect size. Which model to use in this case, is determined from the observed data based on heterogeneity measures, which are calculated under the assumption of a fixed-effects model. If heterogeneity is nonsignificant, a fixed-effects model is an appropriate characterization of data. Otherwise, a random-effects model best characterizes the data. The results from such *conditional* approaches should, however, not be confounded with statements regarding population parameters.

We conducted our analysis from both an unconditional and a conditional perspective. For the unconditional perspective, we chose the random-effects model. Hedges and Vevea state that "In the case of random-effects models, for example, some individual effect-size parameters may be negative even though μ is positive. That corresponds to the substantive idea that some realizations of the treatment may actually be harmful even if the average effect is beneficial" [27]. Results in [1] suggest that the effects of pair programming may be positive or negative dependent on other factors (e.g., expertise and task complexity). Also, the outcome constructs *Quality*, *Duration*, and *Effort* are not yet well-defined in software engineering. These constructs are operationalized in very diverse ways (Section 3.1), and for the time being, it is reasonable to view these diverse operationalizations as different aspects of a construct (a so-called *formative* measurement model [3,6,12,40]). Under these circumstances, and until the constructs are better understood, it is reasonable to relate to a random-effects model.

In the conditional approach, we tested whether there were genuine differences underlying the results of the studies (heterogeneity), or whether the variation in the findings was compatible with chance alone.

In the following, we give an overview of the technical details of the meta-analysis. For further elaborations, see e.g., [27,4,39]. Let k be the number of studies in the meta-analysis. Let T_i be the standardized effect size estimate (in our case, Hedges' g) of study i. In the fixed-effects model, the estimate $\overline{T}_\bullet$ of the assumed fixed population effect size, and the estimate's variance $v_\bullet$, are

$$\overline{T}_\bullet = \frac{\sum_{i=1}^k w_i T_i}{\sum_{i=1}^k w_i} \qquad v_\bullet = \frac{1}{\sum_{i=1}^k w_i} \tag{4}$$

where $w_i = 1/v_i$ is the weight assigned to study i, i.e., the reciprocal of the variance v_i for study i. Thus, $\overline{T}_\bullet$ is a weighted mean over the effect sizes of the individual studies, where studies with less variance are given greater weight. In the random-effects model, the weights are based on between-study variance in addition to within-study variance v_i. Specifically, the estimate $\overline{T}^*_\bullet$ of the mean μ of the assumed population effect size distribution, and the estimate's variance $v^*_\bullet$, are

$$\overline{T}^*_\bullet = \frac{\sum_{i=1}^k w^*_i T_i}{\sum_{i=1}^k w^*_i} \qquad v^*_\bullet = \frac{1}{\sum_{i=1}^k w^*_i} \tag{5}$$

where $w^*_i = 1/v^*_i$, for $v^*_i = v_i + \tau^2$. Here, τ^2 is the additional between-study variance:

$$\tau^2 = \begin{cases} \dfrac{Q - df}{C} & \text{if } Q > df \\ 0 & \text{if } Q \leqslant df \end{cases} \tag{6}$$

where the degrees of freedom $df = k - 1$, and Q represents the total variance:

[1] Comprehensive Meta-Analysis is a trademark of Biostat Inc.

$$Q = \sum_{i=1}^{k} w_i (T_i - \overline{T}_\bullet)^2 \quad (7)$$

In Eq. (6), C is simply a scaling factor that ensures that τ^2 has the same denomination as within-study variance, i.e., $C = \sum w_i - \sum w_i^2 / \sum w_i$.

In fact, Q is a statistic that indicates heterogeneity, and one that we used for this purpose. A significant Q rejects the null hypothesis of homogeneity and indicates that the variability among the effect sizes is greater than what is likely to have resulted from subject-level variability alone [27]. We also calculated the I^2-statistic, which indicates heterogeneity in percentages:

$$I^2 = 100\%(Q - df)/Q \quad (8)$$

A value of 0 indicates no observed heterogeneity, 25% low, 50% moderate, and 75% high heterogeneity [28].

2.7. Assumption of parametric normality

Hedges' g is a parametric effect size measure that is based, firstly, on the assumption that one wishes to relate to a population distribution, and, secondly, that each sub-population (solo, pair) is normally distributed on the response variable. This assumption also underlies the meta-analytic procedures and estimates that we used. It is well-known that *Duration* and *Effort* measures have Gamma distributions, and the various *Quality* measures in this study may or may not have normal distributions. Thus, one may question whether other standardized effect-size measures should have been used for the meta-analysis. Non-parametric effect-size measures would merely allow one to characterize data and not allow one to relate to a population. On the other hand, specifying other distributions (e.g., Gamma) would demand that we had access to the individual studies' raw data. Since means are meaningful measures for all three outcome constructs, we therefore decided to use Hedges' g as a best compromise for conducting the meta-analysis. Note also that even if the assumed population distribution is not evident in a (small) sample, this reason alone should not lead one to abandon the model (unless the sole aim is to characterize data). Small samples will of course, lead to less confident parameter estimates but these confidence estimates will be calculated correctly according the assumed population distribution.

3. Results

We first present characteristics of the 18 studies that were included in the review and meta-analysis. We then give the meta-analytic results in terms of overall effects, subgroup effects and publication bias.

3.1. Description of studies

Characteristics of the 18 studies included in the meta-analysis are summarized in Table 1 in alphabetical order, while full citations are provided in the Appendix.

Of the 18 studies, 11 were from Europe and seven from North America. The number of subjects in the studies varied from 12 to 295. Thirteen of the studies used students as subjects, while four used professionals. One used both professionals and students. Five studies made the comparison within a team context, that is, teams of pairs versus teams of individuals (marked with an asterisk in the first column). The studies often administered several tests and the number of data points may have varied across tests (numbers in parentheses in Table 1).

All studies used programming tasks as the basis for comparison. In addition, Madeyski (2006) and Madeyski (2007) included test-

Table 1
Characteristics of the included studies.

Study	Subjects	N_{Tot}	N_{Pair}	N_{Ind}	Study setting
Arisholm et al. (2007)	Professionals	295	98	99	10 sessions with individuals over 3 months and 17 sessions with pairs over 5 months (each of 1 day duration, with different subjects). Modified 2 systems of about 200–300 Java LOC each
*Baheti et al. (2002)	Students	134	16	9	Teams had 5 weeks to complete a curricular OO programming project. Distinct projects per team
Canfora et al. (2005)	Students	24	12	24	2 applications each with 2 tasks (run1 and run2)
Canfora et al. (2007)	Professionals	18	5(4)	8(10)	Study session and 2 runs (totalling 390 min) involving 4 maintenance tasks (grouped in 2 assignments) to modify design documents (use case and class diagrams)
Domino et al. (2007)	Professionals Students	88	28	32	Run as several sessions during a period of two months. Pseudocode on "Create-Design" tasks Test-driven development
*Heiberg et al. (2003)	Students	84(66)	23(16)	19(17)	4 sessions over 4 weeks involving 2 programming tasks to implement a component for a larger "gamer" system
Madeyski (2006)	Students	188	28	31(35)	8 laboratory sessions involving 1 initial programming task in a finance accounting system (27 user stories)
Madeyski (2007)	Students	98	35	28	Java course project of developing a 27 user story accounting system over 8 laboratory sessions of 90 min each. Test-driven development
Müller (2005)	Students	38	19	23	2 runs of 1 programming session each on 2 initial programming tasks (Polynomial and Shuffle-Puzzle) producing about 150 LOC
Müller (2006)	Students	18(16)	4(5)	6	1 session involving initial design + programming tasks on an elevator system
Nawrocki & Wojciechowski (2001)	Students	15	5	5	4 lab sessions over a winter semester, as part of a University course. Wrote four C/C++ programs ranging from 150–400 LOC
Nosek (1998)	Professionals	15	5	5	45 min to solve 1 programming task (database consistency check script)
*Phongpaibul & Boehm (2006)	Students	95	7	7	12 weeks to complete 4 phases of development + inspection
*Phongpaibul & Boehm (2007)	Students	36	5	4	Part of a team project to extend a system. 13 weeks to complete 4 phases of development + inspection
Rostaher & Hericko (2002)	Professionals	16	6	4	6 small user stories filling 1 day
*Vanhanen & Lassenius (2005)	Students	20	4	8	9-week student project in which each subject spent a total of 100 h (400 h per team). A total of 1500–4000 LOC was written
Williams et al. (2000)	Students	41	14	13	6-week course where the students had to deliver 4 programming assignments
Xu & Rajlich (2006)	Students	12	4	4	2 sessions with pairs and 1 session with individuals. 1 initial programming task producing around 200–300 LOC

Table 2
Summary of meta-analysis.

Analysis	k	Model	Effect Size			Heterogeneity			
			g	95% CI	p	Q	I^2	τ^2	p
Overall effects									
Quality	14	fixed	.23	.09 .37	.00	35.97	63.86	.14	.00
		random	.33	.07 .60	.01				
Duration	11	fixed	.40	.21 .59	.00	33.57	70.21	.28	.00
		random	.54	.13 .94	.01				
Effort	11	fixed	−.73	−.94 −.51	.00	66.36	84.93	.87	.00
		random	−.52	−1.18 .13	.12				
Subgroups Students									
Quality	11	fixed	.22	.06 .38	.01	32.97	69.66	.18	.00
		random	.32	−.01 .65	.06				
Duration	7	fixed	.58	.33 .84	.00	12.28	51.13	.13	.06
		random	.63	.24 1.02	.00				
Effort	8	fixed	−.59	−.88 −.30	.00	48.85	85.67	1.17	.00
		random	.04	−.82 .91	.92				
Professionals									
Quality	3	fixed	.26	−.05 .57	.10	2.97	32.56	.07	.23
		random	.37	−.10 .85	.12				
Duration	4	fixed	.16	−.13 .46	.28	16.87	82.22	.83	.00
		random	.50	−.55 1.54	.35				
Effort	3	fixed	−.90	−1.22 −.58	.00	15.48	87.08	1.54	.00
		random	−1.99	−3.56 −.41	.01				
Teams									
Quality	3	fixed	.19	−.22 .60	.36	.23	.00	.00	.89
		random	.19	−.22 .60	.36				
Duration	2	fixed	.34	−.13 .81	.16	1.46	31.55	.06	.23
		random	.31	−.27 .89	.30				
Effort	2	fixed	.74	−.13 1.61	.09	11.25	91.11	4.14	.00
		random	.99	−1.96 3.94	.51				
No teams									
Quality	11	fixed	.24	.09 .39	.00	35.70	71.99	.18	.00
		random	.38	.05 .70	.02				
Duration	9	fixed	.41	.20 .63	.00	32.02	75.02	.37	.00
		random	.63	.13 1.13	.01				
Effort	9	fixed	−.82	−1.05 −.60	.00	43.36	81.55	.61	.00
		random	−.85	−1.48 −.23	.01				

driven development; Müller (2005), Phongpaibul and Boehm (2006), and Phongpaibul and Boehm (2007) included inspections; and Müller (2006) included design tasks.

Quality was typically reported as the number of test cases passed or number of correct solutions of programming tasks, but student grades, delivered functionality, and metrics for code complexity were also used as measures of *Quality*. *Duration* was reported mainly in two modes: as the total time taken to complete all tasks considered (all solutions), or as the total time taken to complete the tasks that had been assessed as having passed a certain quality standard (checked solutions). 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, e.g., teams of individuals versus teams of pairs, *Effort* was reported as the total effort spent by the respective groups.

Thus, the studies included in the meta-analysis do not all apply the same measures or have similar context variables. Rather, they investigate the effectiveness of pair programming with respect to different aspects of the constructs *Quality*, *Duration*, and *Effort*. As such, the studies may be seen as differentiated replications [36], and any measure from a particular study in the meta-analysis is but one indicator of, perhaps, many indicators regarding one of these constructs.

3.2. Effects of pair programming

Table 2 summarizes the meta-analysis. The g column shows the meta-analytic estimates $\overline{T}_{\bullet}$ and $\overline{T}^{*}_{\bullet}$, in terms of Hedges' g, of the population effect size parameter in the fixed-effects model and random-effects model, respectively, along with 95% confidence intervals and p-values. Also given are heterogeneity measures calculated under the assumption of a fixed-effects model. The *Overall effects* analysis will be described in detail in this section, and the *Subgroup* analysis will be given in detail in Section 3.3.

Overall, 14 studies used *Quality* as an outcome construct, 11 used *Duration*, and 11 used *Effort*. Fig. 1 shows Forest plots[2] of the standardized effects for each of the three outcome constructs. The studies are sorted according to the relative weight that a study's effect size receives in the meta-analysis. Relative weights are normalized versions of the weights w_i or w_i^* used in calculating the meta-analytic estimates $\overline{T}_{\bullet}$ or $\overline{T}^{*}_{\bullet}$. The rightmost columns in Fig. 1 show these weights according to the fixed-effects and random-effects models. Estimates from larger studies will usually be more precise than the estimates from smaller studies; hence, larger studies will generally be given greater weight.

The squares indicate the effect size estimate for each study. The size of each square is proportional to the relative weight of the study according to the random-effects model. The relative weights of a random-effects model are generally more uniform than those of fixed-effects models, due to the incorporation of between-study variance into all the studies' weights. The horizontal lines indicate the 95% confidence intervals for each study's effect size estimate according to the random-effects model.

The diamonds at the bottom give the meta-analytic effect size estimate according to the fixed-effects and the random-effects model, i.e., $\overline{T}_{\bullet}$ and $\overline{T}^{*}_{\bullet}$, respectively. The diamonds' centers and widths indicate the estimates and their 95% confidence interval, respectively.

Fig. 2 shows *one-study-removed* analyses for each of the three outcome constructs. The plots show the meta-analytic effect size estimate when each study is removed from the meta-analysis. The resulting deviation from the full analysis indicates the *sensitivity* of the full analysis with respect to each study, that is, how much difference a given study makes to the meta-analysis.

3.2.1. Effects of pair programming on Quality

Fourteen studies compared the effects of pair programming on *Quality* in a total of 38 tests. These studies used a total of 1160 subjects, although in some studies, not all subjects were used in these particular tests. The subjects were distributed to study units (pairs, solo, teams of pairs, teams of solo) in various ways (Section 3.1). The meta-analytic effect size estimate is .23 in the fixed-effects model and .33 in the random-effects model.

Both the fixed-effects model and the random-effects model suggest that there is a small[3] positive overall effect of pair programming on *Quality* compared with solo programming. Only one study showed a negative effect of pair programming on *Quality* (Domino et al., 2007). All the other studies showed mostly small to medium positive effects. The three studies by Domino et al. (2007), Arisholm et al. (2007), and Madeyski (2006) contribute more than 50% of the total weight in the meta-analysis for *Quality*. The one-study-removed analysis shows that the meta-analysis is most sensitive to the inclusion/exclusion of Williams et al. (2000). Heterogeneity is significant at a medium level ($Q = 35.97$; $p < .01$; $I^2 = 63.86\%$).

3.2.2. Effects of pair programming on Duration

Eleven studies reported effects on *Duration* in a total of 21 tests. These studies used a total of 669 subjects. Both the fixed-effects model and the random-effects model suggest that there is a medium positive overall effect of pair programming on *Duration*.

[2] The plots were generated with PSTricks postscript macros in LaTeX within an Excel spreadsheet using data produced by Comprehensive Meta-Analysis.

[3] The effect size is small compared with the effect sizes reported in other software engineering experiments [30].

Study	g	95% C.I.		p	Relative weight fixed	Relative weight random
Quality						
Domino et al. (2007)	-.27	-.58	.05	.10	20.15	11.31
Arisholm et al. (2007)	.11	-.24	.46	.52	16.31	10.90
Madeyski (2006)	.08	-.28	.43	.67	16.03	10.87
Williams et al. (2000)	1.04	.65	1.43	.00	13.07	10.42
Madeyski (2007)	.17	-.32	.67	.49	8.27	9.21
Heiberg et al. (2003)	.10	-.44	.64	.71	6.88	8.66
Müller (2005)	.28	-.32	.88	.36	5.58	8.00
Canfora et al. (2007)	.69	-.09	1.46	.08	3.32	6.27
Baheti et al. (2002)	.30	-.50	1.09	.46	3.18	6.12
Phongpaibul & Boehm (2006)	.32	-.69	1.32	.54	1.97	4.59
Müller (2006)	.51	-.59	1.62	.36	1.64	4.05
Phongpaibul & Boehm (2007)	.18	-1.00	1.36	.77	1.43	3.68
Nosek (1998)	.91	-.28	2.10	.13	1.42	3.66
Xu & Rajlich (2006)	2.20	.58	3.82	.01	.77	2.26
Overall fixed model	.23	.09	.37	.00		
Overall random model	.33	.07	.60	.01		
Duration						
Arisholm et al. (2007)	.21	-.13	.54	.22	33.33	13.68
Canfora et al. (2005)	.57	.07	1.07	.02	15.24	12.30
Nawrocki & Wojciechowski (2001)	.16	-.40	.73	.57	11.83	11.67
Heiberg et al. (2003)	.55	-.03	1.14	.06	10.97	11.47
Müller (2005)	1.30	.64	1.95	.00	8.66	10.77
Canfora et al. (2007)	-.59	-1.37	.20	.14	6.09	9.61
Baheti et al. (2002)	-.05	-.84	.74	.89	6.01	9.56
Rostaher & Hericko (2002)	.06	-1.08	1.21	.91	2.87	6.82
Müller (2006)	.98	-.24	2.20	.12	2.51	6.33
Xu & Rajlich (2006)	1.85	.34	3.35	.02	1.65	4.85
Nosek (1998)	4.09	1.98	6.20	.00	.84	2.93
Overall fixed model	.40	.21	.59	.00		
Overall random model	.53	.13	.94	.01		
Effort						
Arisholm et al. (2007)	-.68	-1.03	-.34	.00	39.14	12.34
Canfora et al. (2005)	-1.09	-1.61	-.58	.00	17.38	11.84
Müller (2005)	-.25	-.85	.35	.42	12.92	11.54
Nawrocki & Wojciechowski (2001)	-1.25	-1.88	-.61	.00	11.62	11.42
Canfora et al. (2007)	-1.93	-2.88	-.97	.00	5.04	10.03
Vanhanen & Lassenius (2005)	-.49	-1.62	.63	.39	3.66	9.27
Müller (2006)	-.64	-1.83	.54	.29	3.32	9.02
Xu & Rajlich (2006)	-.11	-1.31	1.10	.86	3.18	8.91
Phongpaibul & Boehm (2006)	2.52	1.17	3.88	.00	2.53	8.26
Nosek (1998)	-4.45	-6.70	-2.20	.00	.92	5.09
Phongpaibul & Boehm (2007)	8.11	4.18	12.03	.00	.30	2.28
Overall fixed model	-.73	-.94	-.51	.00		
Overall random model	-.52	-1.18	.13	.12		

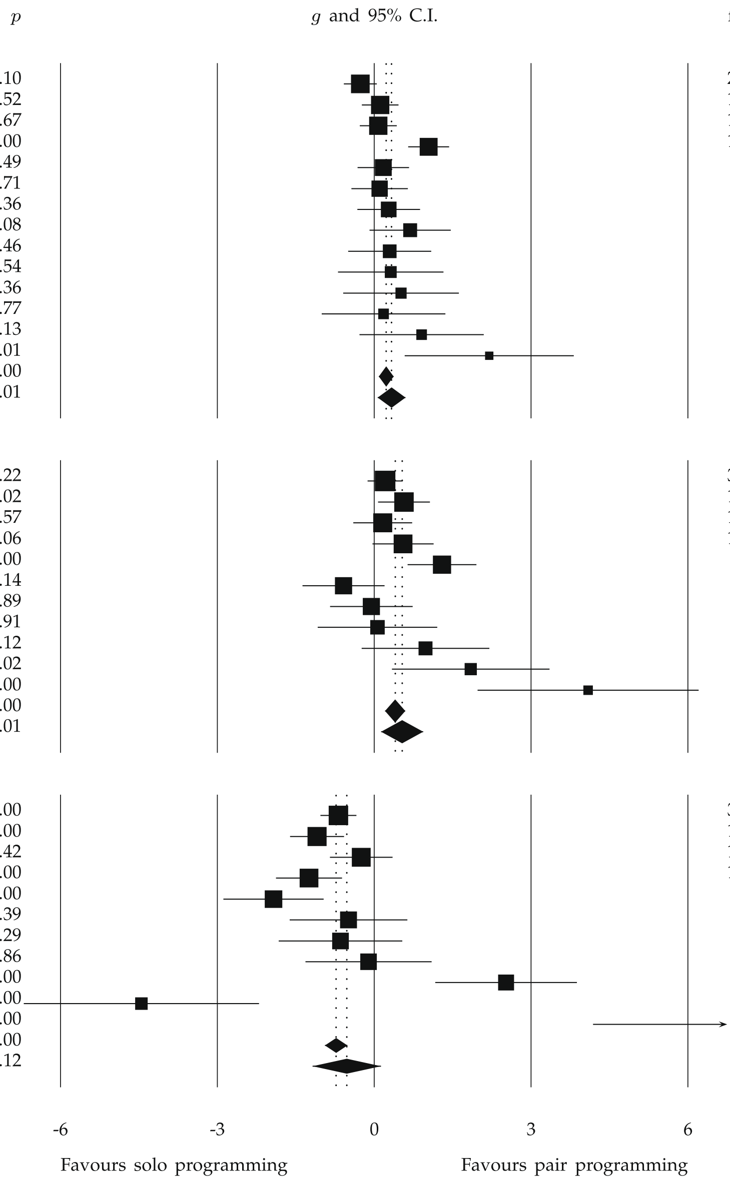

Fig. 1. Forest plots for meta-analysis of *Quality*, *Duration* and *Effort*.

Compared with *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. In addition, the smaller studies show medium to large contradictory effects. The three studies by Arisholm et al. (2007), Canfora et al. (2005), and Nawrocki and Wojciechowski (2001) contribute more than 60% of the total weight in the meta-analysis of *Duration*. The one-study-removed analysis shows that the meta-analysis is most sensitive to the inclusion/exclusion of Nosek (1998). Heterogeneity is significant at a medium level ($Q = 33.57$; $p < .01$; $I^2 = 70.21\%$).

3.2.3. Effects of pair programming on Effort

Eleven studies reported effects on *Effort* in a total of 18 tests. These studies used a total of 586 subjects. Both the fixed-effects model and the random-effects model suggest that there is a medium negative overall effect of pair programming on *Effort* compared with solo programming. All the included studies show a negative effect on *Effort*, apart from the two studies by Phongpaibul and Boehm (2006,2007). However, the results of those studies are not directly comparable, because the researchers compared pair programming teams with teams of individuals who also performed inspections. The three studies by Arisholm et al. (2007), Canfora et al. (2005), and Müller (2005) contribute almost 70% of the total weight in the meta-analysis of *Effort*. The one-study-removed analysis shows that the meta-analysis is most sensitive to the inclusion/exclusion of either of Phongpaibul and Boehm (2006,2007). Heterogeneity is significant and high ($Q = 66.36$; $p < .01$; $I^2 = 84.93\%$).

3.3. Subgroup analyses

Because of medium to high heterogeneity, we decided to conduct subgroup analyses as a step to identify possible immediate sources of heterogeneity. Two subgroup types stand out due to surface dissimilarities: the type of developers (students or professionals) and the type of comparison (isolated pairs vs. individuals, or teams of pairs vs. teams of individuals). The results of these analyses are summarized in Table 2.

The most dramatic result of the subgroup analysis is the reversal of effect on *Effort* for the *Teams* subgroup (from −.52 in the

Study	One study removed g	95% C.I.		p
Quality				
Arisholm et al. (2007)	.37	.07	.67	.02
Baheti et al. (2002)	.34	.06	.62	.02
Canfora et al. (2007)	.31	.03	.59	.03
Domino et al. (2007)	.40	.14	.65	.00
Heiberg et al. (2003)	.36	.07	.65	.01
Madeyski (2006)	.37	.07	.67	.02
Madeyski (2007)	.36	.06	.65	.02
Müller (2005)	.34	.06	.63	.02
Müller (2006)	.33	.05	.60	.02
Nosek (1998)	.31	.04	.58	.02
Phongpaibul & Boehm (2007)	.34	.06	.62	.02
Phongpaibul & Boehm (2006)	.34	.06	.62	.02
Williams et al. (2000)	.17	-.03	.37	.09
Xu & Rajlich (2006)	.28	.03	.54	.03
Overall	.33	.07	.60	.01
Duration				
Arisholm et al. (2007)	.61	.13	1.10	.01
Baheti et al. (2002)	.60	.17	1.04	.01
Canfora et al. (2005)	.55	.09	1.02	.02
Canfora et al. (2007)	.64	.24	1.04	.00
Heiberg et al. (2003)	.55	.09	1.01	.02
Müller (2005)	.42	.03	.82	.04
Müller (2006)	.51	.09	.93	.02
Nawrocki & Wojciechowski (2001)	.60	.14	1.05	.01
Nosek (1998)	.41	.08	.75	.02
Rostaher & Hericko (2002)	.58	.15	1.00	.01
Xu & Rajlich (2006)	.46	.06	.86	.02
Overall	.53	.13	.94	.01
Effort				
Arisholm et al. (2007)	-.45	-1.29	.39	.30
Canfora et al. (2005)	-.42	-1.19	.34	.28
Canfora et al. (2007)	-.37	-1.05	.31	.29
Müller (2005)	-.54	-1.28	.20	.15
Müller (2006)	-.50	-1.21	.20	.16
Nawrocki & Wojciechowski (2001)	-.42	-1.15	.32	.27
Nosek (1998)	-.35	-.97	.28	.28
Phongpaibul & Boehm (2007)	-.72	-1.29	-.15	.01
Phongpaibul & Boehm (2006)	-.82	-1.40	-.25	.01
Vanhanen & Lassenius (2005)	-.52	-1.23	.19	.15
Xu & Rajlich (2006)	-.56	-1.26	.14	.12
Overall	-.52	-1.18	.13	.12

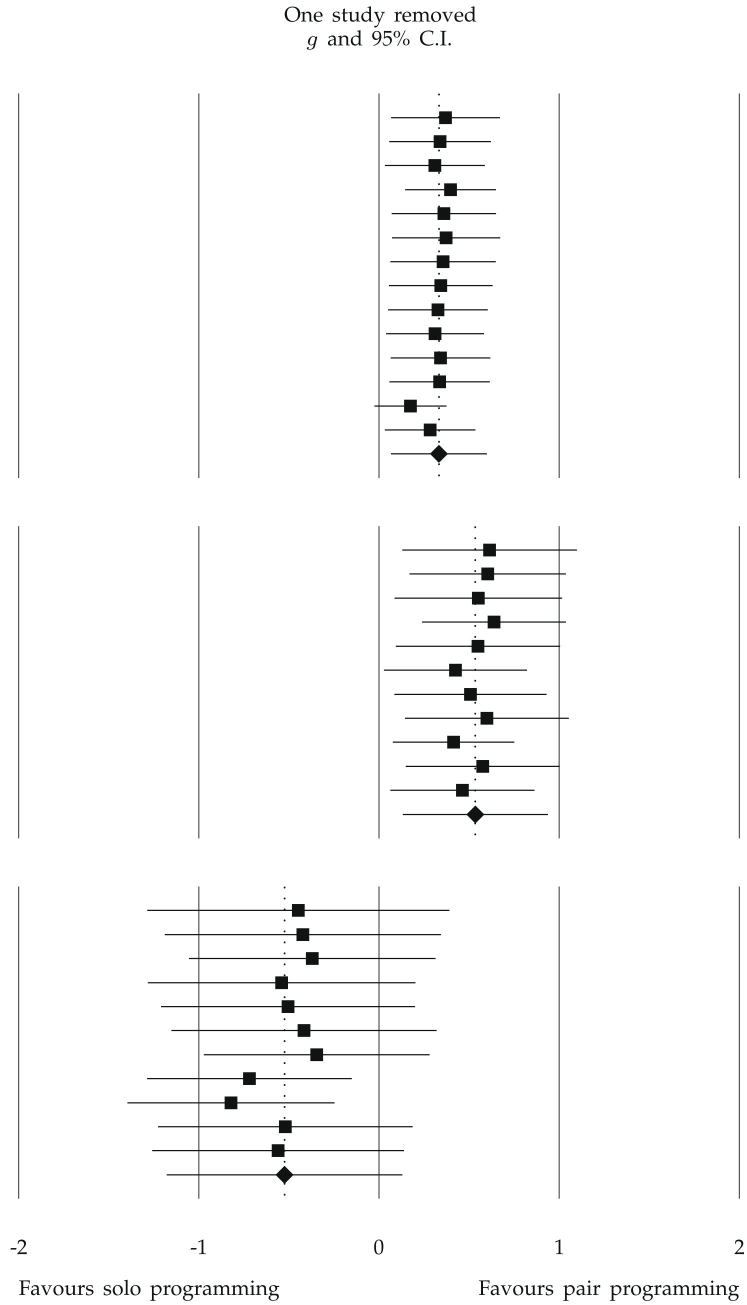

Fig. 2. Forest plots for one-study-removed meta-analysis of *Quality*, *Duration* and *Effort*.

overall analysis to .99 (non-significant) in the subgroup analysis, for the random-effects model). This is due to the two studies Phongpaibul and Boehm (2006, 2007), which we remarked upon above. Without these two studies (*No teams* subgroup), the effect size increases to from .−52 to −.85 in the random-effects model.

Apart from this reversal, effect sizes in the subgroups remain in the same order of magnitude as in the overall analysis, except for the following cases: For *Quality*, the *No teams* subgroup reaches a medium level in the random model's estimate (.38). For *Duration*, the *Professionals* subgroup declines to a small level in the fixed model's estimate (.16). For *Effort*, the *Students* subgroup declines to small (.04) (but non-significant, $p = .92$), and the *Professionals* subgroup increases to a large effect size (−1.99) in their respective random-effects models.

Heterogeneity levels remain medium to high. *Professionals* and *No teams* increase to high heterogeneity for *Duration* (82.22) and *Teams* decreases to small (.00) (but non-significant, $p = .89$) for *Quality*.

Significance levels for both effect size estimates and heterogeneity decrease dramatically in several instances in the subgroups, probably due to small k. This is particularly the case in the *Professionals* and *Teams* subgroups. Note however, that the effect on *Effort* actually turns significant in the *Professionals* subgroup. Note that the *Professionals* subgroup excludes Phongpaibul and Boehm (2006, 2007) since they have student subjects.

3.4. Publication bias

Publication bias captures the idea that studies that report relatively large treatment effects are the ones that were most likely to be initiated in the first place, and/or submitted and accepted for publication. The effect size estimated from a biased collection of studies will tend to overestimate the true effect. We used Comprehensive Meta-Analysis to assess the likely extent of publication bias, and its potential impact on the conclusions.

Figs. 3–5 show funnel plots for *Quality*, *Duration* and *Effort*, respectively. These graphs plot each study according to a measure of study size called *precision* (which is simply 1/StdError) on the vertical axis, and according to effect size (Hedges' *g*) on the horizontal axis.

Hence, large studies tend to appear toward the top of the graph, and smaller studies tend toward the bottom of the graph. Since there is more inter-study variance in effect size estimates among the smaller studies, these studies will be dispersed wider, while larger studies will tend to cluster near the meta-analytic estimate of effect size.

In the absence of publication bias, studies should be distributed symmetrically on either side of the meta-analytic estimate of effect size. In the presence of bias, the bottom part of the plot should show a higher concentration of studies on one side of the meta-analytic estimate than the other. This would reflect the fact that smaller studies are more likely to be published if they have larger than average effects, which makes them more likely to meet the criterion for statistical significance. Various statistical measures can be used to complement the picture given by the funnel plots.

Duval and Tweedie's *trim and fill* method [13,14], takes the basic idea behind the funnel plot one step further and imputes (computes and inserts) assumed missing studies to obtain symmetry if the funnel plot is asymmetric and then recomputes the meta-analytic estimate of effect size. The method initially trims the asymmetric studies from the biased side to locate the unbiased effect (in an iterative procedure), and then fills the plot by re-inserting the trimmed studies on the biased side as well as their imputed counterparts on the opposite side of the meta-analytic estimate of effect size. Figs. 6–8 show the plots of Figs. 3–5 with imputed studies (black dots) and adjusted meta-analytic estimates of effect size (black diamonds).

These analyses suggest that there is indeed some publication bias in our sample. For example, six studies are imputed to obtain symmetry for *Quality*. This might be taken as an incentive to publish more high-quality studies that, perhaps, exhibit low effects or opposite effects to those that are expected. This might include promoting gray literature to be readily accessible in journals and conferences.

However, it is important to note that publication bias may not be the only cause of funnel plot asymmetry. For instance, asymmetry may be caused by between-study heterogeneity, study quality, or other factors [44].

In the presence of heterogeneity (which we are assuming), there are three meaningful ways to impute the assumed missing studies. (1) A random-effects model is used to trim and fill, and then the adjusted meta-analytic estimate of effect size is calculated from the filled data using a random-effects model (a so-called *random–random* approach) [13,14]. However, in a random-effects model meta-analysis, smaller studies are given added weight in the synthesis. Thus, if publication bias exists, the meta-analytic estimate is likely to be more biased than that obtained from a fixed-effects model meta-analysis [44]. Thus, an alternative is: (2) a fixed-effects model is used to trim and fill, and then the adjusted meta-analytic estimate of effect size is calculated from the filled data using a fixed-effects model (*fixed–fixed*). It is recommended to report both fixed–fixed and random–random analyses [51]. The fixed–fixed approach may be unsatisfying if one wishes to view adjustments relative to the original random-effects model meta-analysis. (3) A third approach is to use a fixed-effects model to trim and fill the meta-analysis, but a random-effects model to calculate the meta-analytic effect size estimate from the filled data (*fixed–random*). The fixed-effects trim and fill process is less likely to be influenced by any publication bias than the random-effects model approach, but the resulting estimate from the random-effects model is likely to be more conservative than if a fixed-effects model is used [44]. The plots in Figs. 6–8 were generated using the fixed–random approach.

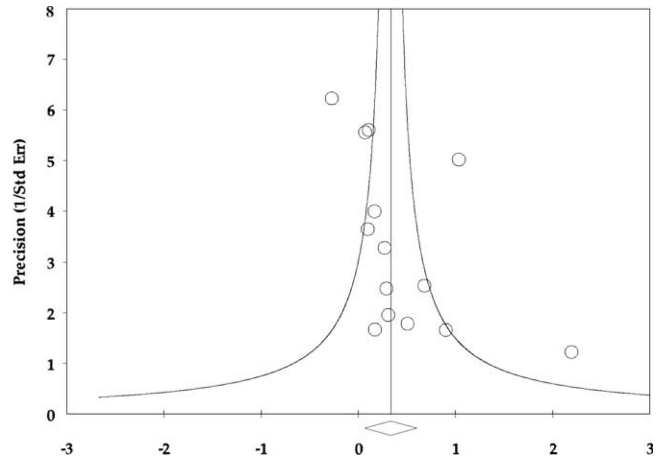

Fig. 3. Funnel plot quality.

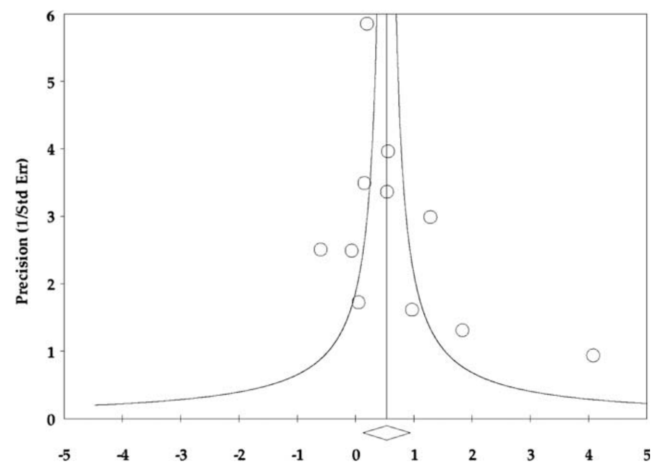

Fig. 4. Funnel plot duration.

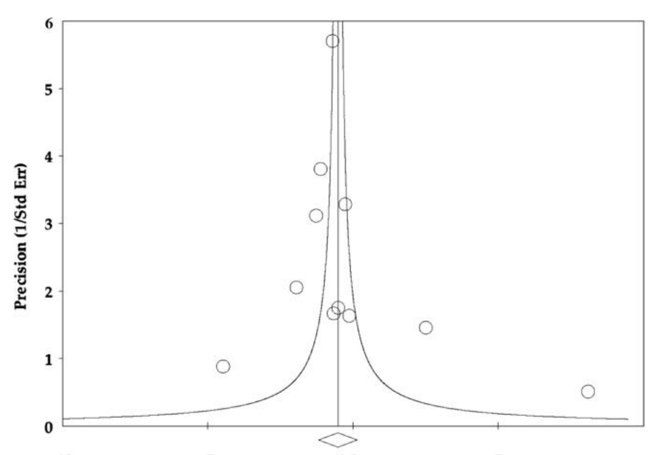

Fig. 5. Funnel plot effort.

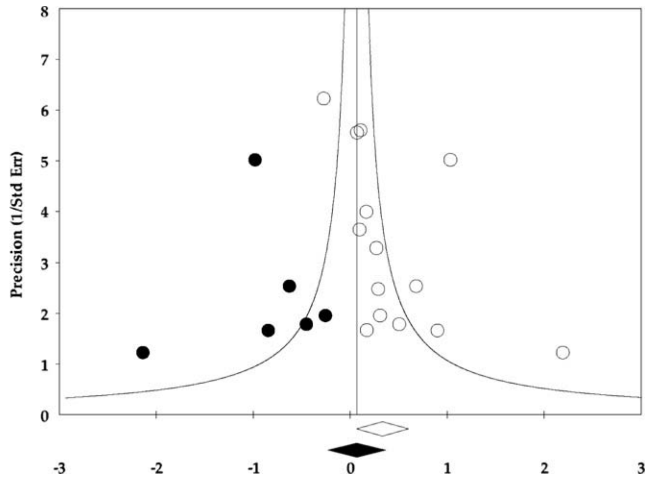

Fig. 6. Funnel plot quality trim and fill fixed–random.

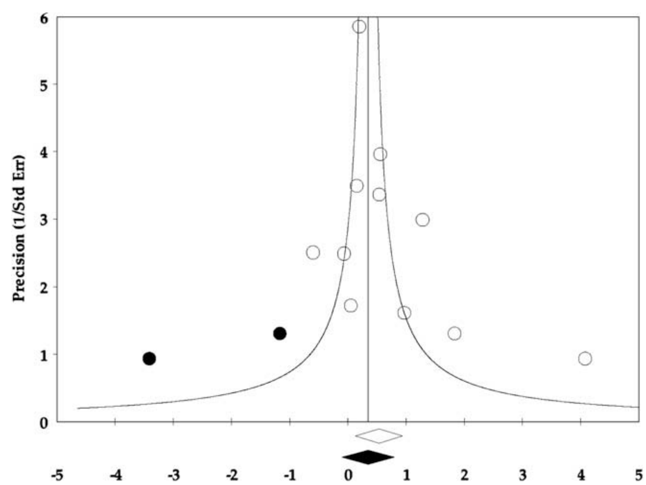

Fig. 7. Funnel plot duration trim and fill fixed–random.

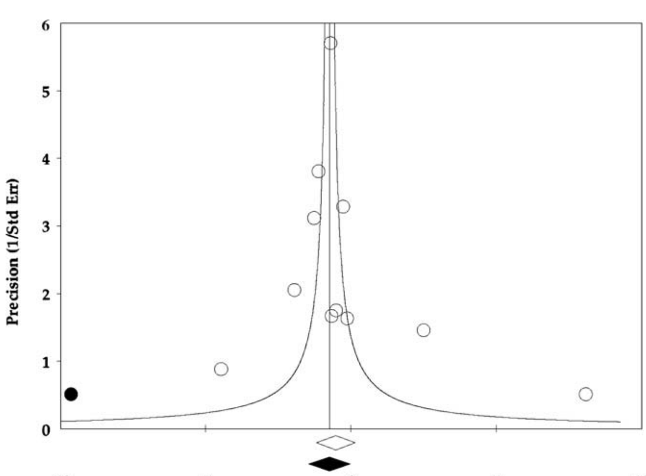

Fig. 8. Funnel plot effort trim and fill fixed–random.

Table 3 presents the results from all perspectives. For example, for *Quality* a fixed-effects model approach to trim and fill yields six imputed studies and adjusts the estimate in the fixed-effects model from .23 to .03 (fixed–fixed), while it adjusts the estimate in the random-effects model from .33 to .07 (fixed–random).

Table 3
Trim and fill analysis.

	Imputed	Fixed		Random		*Q*	
		g	95% C.I.	*g*	95% C.I.		
Quality							
Observed		.23	.09 .37	.33	.07 .60	35.97	
fixed	6	.03	−.09 .16	.07	−.23 .37	81.42	Left
random	0	.23	.09 .37	.33	.07 .60	35.97	Left
Duration							
Observed		.40	.21 .59	.53	.13 .94	33.57	
fixed	2	.34	.15 .54	.35	−.10 .80	49.77	Left
random	1	.37	.18 .56	.44	−.01 .88	45.01	Left
Effort							
Observed		−.73	−.94 −.51	−.52	−1.18 .13	66.36	
fixed	1	−.75	−.97 −.54	−.74	−1.46 −.01	86.01	Left
random	2	−.58	−.78 −.37	−.07	−.79 .65	104.46	Right

The trim and fill procedure searches for missing studies on one side at a time. For *Qualtiy* and *Duration* missing studies were found on the left (Table 3), i.e., in the opposite direction of the estimated effect, which is consistent with the assumption of publication bias. For *Effort*, the fixed approach found a missing study on the left, while the random approach found missing studies on the right. The missing study on the left is not consistent with the assumption of publication bias and is due to the overly large effort estimate of one of the studies in the opposite direction of the estimate.

Sterne and Egger [50] note that trim and fill plots merely detect a relationship between sample size and effect size, not a causal mechanism between the two. The effect size may be larger in small studies due to publication bias. However, it is also possible that small studies estimate a different part of the population effect size distribution than do large studies, e.g., because smaller studies use different operationalizations of the outcome constructs, or have different situational variables than large studies. Because one does not know for certain whether funnel plot asymmetry is really caused by publication bias, it is recommended to use the trim and fill method mainly as a form of sensitivity analysis [44].

3.5. Moderating effects of task complexity and expertise

Due to the interdisciplinary and complex nature of industrial software engineering, it is usually not reasonable to test an hypothesis that considers only one independent (predictor) variable and one dependent (criterion) variable. Hypotheses related to software engineering should typically include additional variables and test more complex relationships in order to provide a more accurate description of reality. Indeed, the relatively small overall effects and large between-study variance (heterogeneity) indicate that one or more moderator variables might play a significant role.

Only two of the studies in the review, Vanhanen and Lassenius (2005) and Arisholm et al. (2007), tested explicitly for moderator effects, while only one of the other studies discussed the potential influence of such effects; Williams et al. (2000) suggested that the relative improvement by pairs after the first programming task in their experiment was due to “pair jelling”.

The literature on group dynamics (e.g., [5,21]) suggests that the extent to which group performance exceeds that of individuals, and the mechanisms by which such gains in performance may be achieved, depend upon the composition of the group and the characteristics of the tasks. Vanhanen and Lassenius (2005) found that task complexity did not affect the differences in effort between solo and pair programming; the Pearson correlation between task complexity and effort difference was as low as $r = -.02$.

On the other hand, Arisholm et al. (2007) found moderating effects of both task complexity and expertise. The results are shown in Fig. 9. Overall, the results showed that the pairs had an 8% decrease in duration ($g = .21$) with a corresponding 84% increase in effort ($g = -.68$) and a 7% increase in correctness ($g = .11$) (Fig. 9a). However, the main effects of pair programming were masked by the moderating effect of system complexity, in that simpler designs had shorter duration, while more complex designs had increased correctness (Fig. 9e).

Furthermore, when considering the moderating effect of programmer expertise, junior pairs had a small (5%) increase in duration and thus a large increase in effort (111%), and a 73% increase in correctness (Fig. 9b). Intermediate pairs had a 28% decrease in duration (43% increase in effort) and a negligible (4%) increase in correctness (Fig. 9c). Senior pairs had a 9% decrease in duration (83% increase in effort) and an 8% decrease in correctness (Fig. 9d). Thus, the juniors benefited from pair programming in terms of increased correctness, the intermediates in terms of decreased duration, while there were no overall benefits of pair programming for seniors. When considering the combined moderating effect of system complexity and programmer expertise on pair programming, there appears to be an interaction effect: Among the different treatment combinations, junior pairs assigned to the complex design had a remarkable 149% increase on correct-

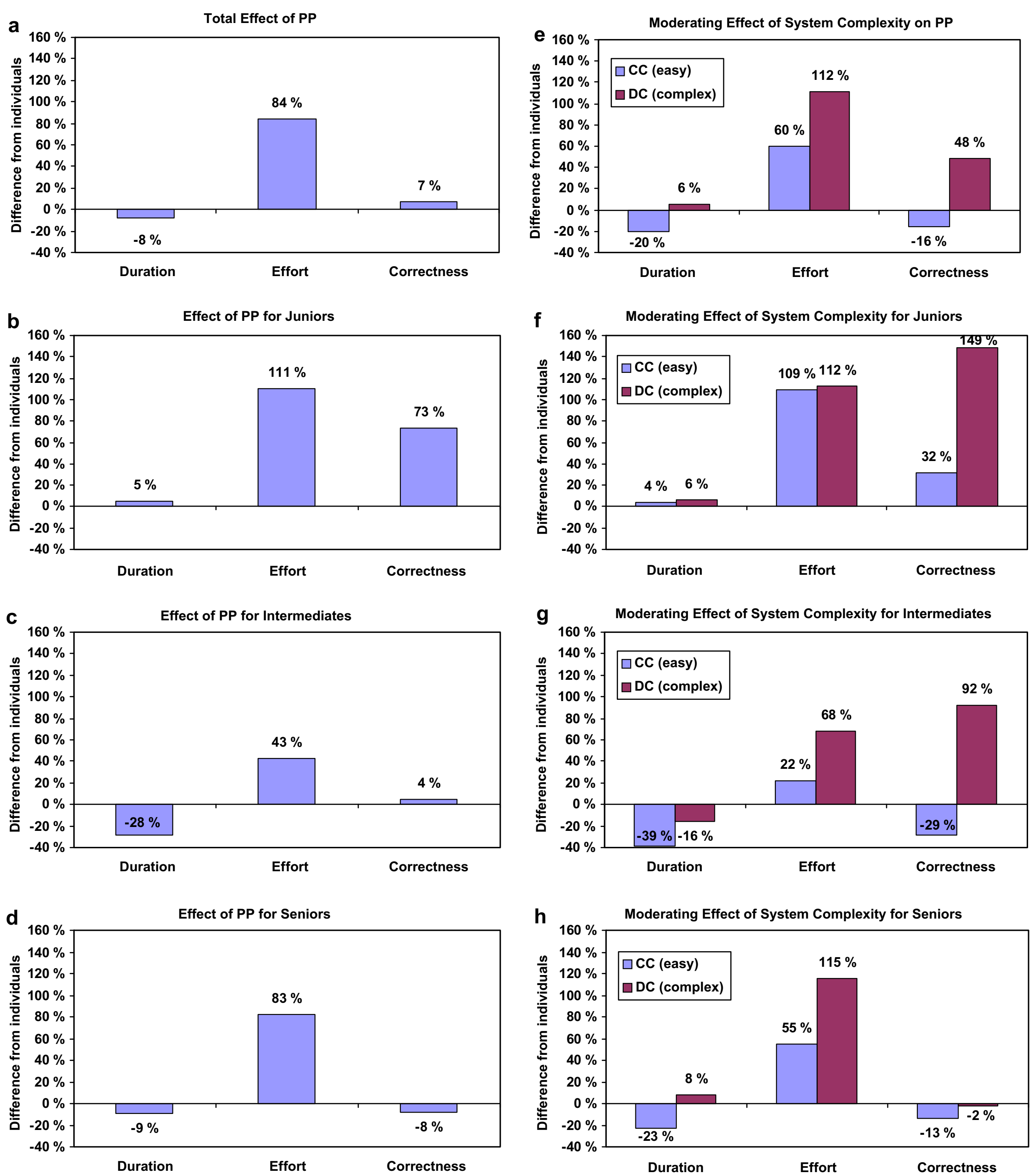

Fig. 9. The moderating effects of programmer expertise (a)–(d) and system complexity (e)–(h) on the relation of pair programming on duration, effort, and correctness (Arisholm et al., 2007).

ness compared with individuals (Fig. 9f). Furthermore, intermediates and seniors experienced an effect of pair programming on duration on the simpler design, with a 39% (Fig. 9g) and 23% (Fig. 9h) decrease, respectively. However, the cost of this shorter duration was a corresponding decrease in correct solutions by 29% and 13%, respectively.

4. Discussion

The unconditional part of the meta-analysis suggests that the population effect size mean μ for each of the outcome constructs are small to medium.

More interestingly, the conditional part of the meta-analysis showed large and partly contradictory differences in the reported overall effects of pair programming, specifically with respect to *Duration* and *Effort*. Our subgroup analyses do not suggest that these differences are due to differences in samples (e.g., students or professionals). Differences in organization (e.g., teams or no teams) were apparent from the subgroup analysis, but these differences were due to two particular studies that, when removed, did not decrease overall heterogeneity.

At the outset, we anticipated between-study variance (heterogeneity) due to moderating factors such as expertise and task complexity and to the fact that the outcome constructs of the meta-analysis are operationalized by indicators that are different aspects of the constructs. Such heterogeneity was taken into account by using a random-effects model for the unconditional interpretation.

Nevertheless, contradictory differences still manifested themselves. Thus, it seems clear that moderating factors play an important role and should be investigated further. The study in [1] corroborates this conclusion for expertise and task complexity, but other moderating factors, such as amount of training in pair programming, motivation, team climate, etc., are likely to be relevant as well.

Consequently, the question of whether pair programming is better than solo programming is not precise enough to be meaningful, since the answer to that question in the present context is both "yes" and "no". On the basis of the evidence from this review, the answer is that "it depends": It depends on other factors, for example, the expertise of the programmers and on the complexity of the system and tasks to be solved.

In the literature, expertise and task complexity are perhaps the most central situation-independent predictors of performance. (Situation-dependent factors, on the other hand, include more dynamic factors such as motivation, team climate, organizational issues, etc.). Theory predicts that experts perform better on complex tasks than do novices because experts' level of understanding corresponds to the deep structure [10,8,9,19] of a complex task. That is, experts perceive (objectively) complex tasks as subjectively less complex, i.e., less *variable* and more *analyzable* [43,38]. Conversely, experts perceive (objectively) less complex tasks as more *variable* and less *analyzable* since such tasks do not match the expert's deep-level understanding of the problem. Novices, on the other hand, do have an understanding that matches the surface structure of a non-complex task, and are expected to do better on non-complex tasks than on complex tasks. Moreover, they may be expected to even outperform experts on non-complex tasks [23].

These effects are evident in [1], in which the levels of correctness for individual juniors, intermediates, and seniors on the non-complex system were 63%, 87%, and 86%, respectively, whereas the levels on the complex system were 34%, 41%, and 81%, respectively. Thus, the performance drop was much higher for juniors than for intermediates and seniors, when moving from the non-complex to the complex system, although juniors did not outperform higher expertise groups on the non-complex system, see also [23].

In our context, one of the most interesting observations is that the pairing up of individuals seems to elevate the junior pairs up to near senior pair performance. Thus, pair collaboration might compensate for juniors' lack of deep understanding, for example, by inducing an expert-like strategy.

Change tasks rely heavily on program comprehension. To comprehend code, experts use a top–down model [47] that short-cuts the available information by only investigating details as dictated by domain knowledge. This approach to comprehension is more efficient than bottom–up comprehension, which builds understanding from details, and which is the approach found to be used by programmers encountering totally unfamiliar code [41,42].

By forcing junior peers to rationalize their ideas to each other, junior pairs might adopt a top–down strategy to comprehension, rather than getting lost in code on their own. The mere act of thinking aloud whilst solving problems has been shown to increase performance, when the verbalization is intended to reason or explain action, e.g., [7] (Type 3 verbalization, in Ericsson and Simon's terminology) [20]. Several studies have concluded that apparent successes of pair programming are not due to the particularities of pair programming (such as the specific roles of driver and navigator), but rather to the shear amount of verbalization that the pair programming situation necessitates [11,22].

Group performance not only relies on task complexity but also on the collaborative nature of a task. In fact, the appropriateness of each of the familiar adages "two heads are better than one", "many hands make light work", and "a chain is only as strong as its weakest link" depends on whether a task is additive, compensatory, disjunctive or conjunctive [48]. For example, the chain analogy is appropriate for conjunctive tasks, where all group members must contribute to the solution, but is inappropriate for disjunctive tasks for which it suffices that one group member has the ability to complete the task. It is not obvious what sort of task pair programming is in this respect.

The precise collaborative nature of pair programming also influences what social mechanisms (social loafing, social labouring, social facilitation, social inhibition, social compensation, etc.) are applicable. However, these social mechanisms also depend on a host of other factors. In a meta-analysis of social loafing (the phenomenon that individuals tend to expend less effort when working collectively than when working individually), Karau and Williams [32] identified several conditions in which such loafing is eliminated (e.g., by high group cohesion) and some in which the opposite phenomenon, social laboring [5], could be observed (i.e., greater effort on group tasks). Social laboring seems to occur when complex or highly involving tasks are performed, or when the group is considered important for its members, or if the prevailing values favor collectivism rather than individualism [5].

5. Conclusion

Our meta-analysis suggests that pair programming is not uniformly beneficial or effective, that inter-study variance is high, and that perhaps publication bias is an issue. Hence, if further investigations are to be undertaken on pair programming, then the focus should be on untangling the moderating factors of the effect of pair programming.

However, with respect to the central factors expertise and task complexity, the current state of knowledge suggest that pair programming is beneficial for achieving correctness on highly complex programming tasks. Pair programming may also have a time gain on simpler tasks. By cooperating, programmers may complete tasks and attain goals that would be difficult or impossible if they worked individually. Junior pair programmers, for example, seem able to achieve approximately the same level of correctness in about the same amount of time (duration) as senior individuals. However, the higher quality for complex tasks comes at a price of a considerably higher effort (cost), while the reduced completion time for the simpler tasks comes at a price of a noticeably lower quality. This fact confirms Voas' [52] contention that you cannot expect faster and better and cheaper. These relationships give rise to a few evidence-based guidelines for the use of pair programming for professional software developers. If you do not know the seniority or skill levels of your programmers, but do have a feeling for task complexity, then employ pair programming either when task complexity is low and time is of the essence, or when task complexity is high and correctness is important.

In the future, we intend to investigate deeper into the theoretical and empirical underpinnings of collaboration in pairs, e.g., by studying group dynamics and analyzing pair dialogues to obtain insights into subjects' learning and reasoning processes. Only by understanding what makes pairs work, and what makes pairs less efficient, can steps be taken to provide beneficial conditions for work and to avoid detrimental conditions; or to avoid pairing altogether when beneficial conditions cannot be provided.

Acknowledgement

The authors are grateful to the anonymous referees for insightful remarks.

Appendix

The following articles were included in the meta-analysis:

Arisholm et al. (2007). E. Arisholm, H. Gallis, T. Dybå, D.I.K. Sjøberg, Evaluating pair programming with respect to system complexity and programmer expertise, IEEE Trans. Software Eng. 33 (2007) 65–86 (Feb.)

Baheti et al. (2002). P. Baheti, E. Gehringer, D. Stotts, Exploring the efficacy of distributed pair programming, in: Proc. XP/Agile Universe 2002, ser. Lecture Notes in Computer Science, vol. 2418, Springer Verlag, 2002, pp. 208–220.

Canfora et al. (2007). G. Canfora, A. Cimitlie, F. Garcia, M. Piattini, C.A. Visaggio, Evaluating performances of pair designing in industry, *J. Systems and Software* 80(8) (2007) 1317–1327 (Aug.)

Canfora et al. (2005). G. Canfora, A. Cimitlie, C.A. Visaggio, Empirical study on the productivity of the pair programming, in: Proc. XP 2005, ser. Lecture Notes in Computer Science,vol. 3556, Springer Verlag, 2005, pp. 92–99.

Domino et al. (2007). M.A. Domino, R. Webb Collins, A.R. Hevner, Controlled experimentation on adaptations of pair programming, Information Technology and Management 8(4) (2007) 297–312 (Dec.)

Heiberg et al. (2003). S. Heiberg, U. Puus, P. Salumaa, A. Seeba, Pair-programming effect on developers productivity, in: Proc. XP 2003, ser. Lecture Notes in Computer Science, vol. 2675. Springer Verlag, 2003, pp. 215–224.

Madeyski (2006). L. Madeyski, The impact of pair programming and test-driven development on package dependencies in object-oriented design – an experiment, in: Proc. PROFES 2006, ser. Lecture Notes in Computer Science, vol. 4034, Springer Verlag, 2006, pp. 278–289.

Madeyski (2007). L. Madeyski, On the effects of pair programming on thoroughness and fault-finding effectiveness of unit tests, in: Proc. PROFES 2007, ser. Lecture Notes in Computer Science, vol. 4589. Springer Verlag, 2007, pp. 207–221.

Müller (2005). M. Müller, Two controlled experiments concerning the comparison of pair programming to peer review, *J. Systems and Software* 78(2) (2005) 169–179

Müller (2006). M. Müller, A preliminary study on the impact of a pair design phase on pair programming and solo programming, *Information and Software Technology* 48(5) (2006) 335–344 (May)

Nawrocki and Wojciechowski (2001). J. Nawrocki, A. Wojciechowski, Experimental evaluation of pair programming, in: Proc. European Software Control and Metrics Conference (ESCOM'01), 2001, pp. 269–276.

Nosek (1998). J. Nosek, The case for collaborative programming, Comm. ACM 41(3) (1998) 105–108 (Mar.)

Phongpaibul and Boehm (2006). M. Phongpaibul, B. Boehm, An empirical comparison between pair development and software inspection in Thailand, in: Proc. Int'l Symposium on Empirical Software Engineering (ISESE'06), 2006, pp. 85–94.

Phongpaibul and Boehm (2007). M. Phongpaibul, B. Boehm, A replicate empirical comparison between pair development and software development with inspection, in: Proc. 1st Int'l Symp. Empirical Software Engineering and Measurement (ESEM'07), IEEE Computer Society, 2007, pp. 265–274.

Rostaher and Herick (2002). M. Rostaher, M. Hericko, Tracking test first pair programming – an experiment, in: Proc. XP/Agile Universe 2002, ser. Lecture Notes in Computer Science, vol. 2418. Springer Verlag, 2002, pp. 174–184.

Vanhanen and Lassenius (2005). J. Vanhanen, C. Lassenius, Effects of pair programming at the development team level: an experiment, in: Proc. Int'l Symposium on Empirical Software Engineering (ISESE'05), 2005, pp. 336–345.

Williams et al. (2000). L. Williams, R.R. Kessler, W. Cunningham, R. Jeffries, Strengthening the case for pair programming, IEEE Software 17(4) (2000) 19–25

Xu and Rajlich (2006). S. Xu, V. Rajlich, Empirical validation of test-driven pair programming in game development, in:Proc. Int'l Conference on Computer and Information Science and Int'l Workshop on Component-Based Software Engineering, Software Architecture and Reuse (ICIS-COMSAR'06), 2006, pp. 500–505.

References

[1] E. Arisholm, H. Gallis, T. Dybå, D.I.K. Sjøberg, Evaluating pair programming with respect to system complexity and programmer expertise, IEEE Trans. Software Eng. 33 (Feb.) (2007) 65–86.
[2] K. Beck, C. Andres, Extreme Programming Explained: Embrace Change, second ed., Addison-Wesley, 2003.
[3] K. Bollen, R. Lennox, Conventional wisdom on measurement: a structural equation perspective, Psychol. Bull. 110 (2) (1991) 305–314.
[4] M. Borenstein, L.V. Hedges, H. Rothstein, Meta-analysis – fixed effects versus random effects, Biostat Inc., Tech. Rep., 2007, <meta-analysis.com/downloads/Meta-analysis_fixed_effect_vs_random_effects.pdf>.
[5] R. Brown, Group Processes: Dynamics within and between Groups, second ed., Blackwell, 2000.
[6] C. Burke jarvis, S.B. Mackenzie, P.M. Podsakoff, A critical review of construct indicators and measurement model misspecification in marketing and consumer research, J. Consum. Res. 30 (Sept.) (2003) 199–218.
[7] M.T.H. Chi, N. de Leeuw, M.H. Chiu, C. LaVancher, Eliciting self-explanations improves understanding, Cognitive Sci. 18 (1994) 439–477.
[8] M.T.H. Chi, P.J. Feltovich, R. Glaser, Categorization and representation of physics problems by experts and novices, Cognitive Sci. (1981) 121–152.
[9] M.T.H. Chi, R. Glaser, E. Rees, Expertise in problem solving, in: R. Sternberg (Ed.), Advances in the Psychology of Human Intelligence, Lawrence Erlbaum Associates Inc., 1982, pp. 17–76.
[10] N. Chomsky, Syntactic Structures, second ed., Mouton, 2002.
[11] J. Chong, T. Hurlbutt, The social dynamics of pair programming, in: Proc. 29th Int'l Conf. Software Engineering, 2007.
[12] A. Diamantopoulos, J.A. Siguaw, Formative versus reflective indicators in organizational measure development: a comparison and empirical illustration, Brit. J. Manag. 17 (2006) 263–282.
[13] S. Duval, R. Tweedie, A nonparametric "trim and fill" method of accounting for publication bias in meta-analysis, J. Am. Stat. Assoc. 95 (445) (2000) 89–98.
[14] S. Duval, R. Tweedie, Trim and fill: a simple funnel-plot-based method of testing and adjusting for publication bias in meta-analysis, Biometrics 56 (2) (2000) 455–463.

[15] T. Dybå, E. Arisholm, D.I.K. Sjøberg, J.E. Hannay, F. Shull, Are two heads better than one? on the effectiveness of pair programming, IEEE Software, November/December 2007, pp. 12–15.
[16] T. Dybå, T. Dingsøyr, G.K.S. Hanssen, Applying systematic reviews to diverse study types: an experience report, in: Proc. 1st Int'l Symp. Empirical Software Engineering and Measurement (ESEM'07), IEEE Computer Society, 2007, pp. 225–234.
[17] T. Dybå, V.B. Kampenes, D.I.K. Sjøberg, A systematic review of statistical power in software engineering experiments, J. Inform. Software Technol. 48 (2006).
[18] T. Dybå, B.A. Kitchenham, M. Jørgensen, Evidence-based software engineering for practitioners, IEEE Software 22 (2005) 58–65.
[19] K.A. Ericsson, N. Charness, Expert performance – its structure and acquisition, Am. Psychol. 49 (1994) 725–747.
[20] K.A. Ericsson, H.A. Simon, Protocol Analysis, revised ed., The MIT Press, 1993.
[21] D.R. Forsyth, Group Dynamics, fourth ed., Thomson Wadsworth, 2006.
[22] S. Freudenberg (née Bryant), P. Romero, B. du Boulay, 'Talking the talk': is intermediate-level conversation the key to the pair programming success story? in: Proc. AGILE 2007, 2007.
[23] T. Hærem, D. Rau, The influence of degree of expertise and objective task complexity on perceived task complexity and performance, J. Appl. Psychol. 92 (5) (2007) 1320–1331.
[24] J.E. Hannay, M. Jørgensen, The role of deliberate artificial design elements in software engineering experiments, IEEE Trans. Software Eng. 34 (Mar/Apr) (2008) 242–259.
[25] J.E. Hannay, D.I.K. Sjøberg, T. Dybå, A systematic review of theory use in software engineering experiments, IEEE Trans. Software Eng. 33 (Feb.) (2007) 87–107.
[26] L.V. Hedges, I. Olkin, Statistical Methods for Meta-Analysis, Academic Press Inc., 1985.
[27] L.V. Hedges, J.L. Vevea, Fixed- and random-effects models in meta-analysis, Psychol. Meth. 3 (4) (1998) 486–504.
[28] J.P. Higgins, S.G. Thompson, J.J. Deeks, D.G. Altman, Measuring inconsistency in meta-analyses, Brit. Med. J. 327 (4) (2003) 557–560.
[29] J.E. Hunter, F.L. Schmidt, Fixed effects vs. random effects meta-analysis models: implications for cumulative research knowledge, Int. J. Select. Assess. 327 (8) (2000) 272–292.
[30] V.B. Kampenes, T. Dybå, J.E. Hannay, D.I.K. Sjøberg, A systematic review of effect size in software engineering experiments, Inform. Software Technol. 49 (11–12) (2007) 1073–1086.
[31] V.B. Kampenes, T. Dybå, J.E. Hannay, D.I.K. Sjøberg, A systematic review of quasi-experiments in software engineering, Inform. Software Technol. 49 (11–12) (2007) 1073–1086.
[32] S.J. Karau, K.D. Williams, Social loafing: a meta-analytic review and theoretical integration, J. Pers. Soc. Psychol. 65 (4) (1993) 681–706.
[33] N.L. Kerr, R.S. Tindale, Group performance and decision making, Ann. Rev. Psychol. 55 (2004) 623–655.
[34] B.A. Kitchenham, S. Charters, Guidelines for Performing Systematic Literature Reviews in Software Engineering, Keele University, EBSE Technical Report, EBSE-2007-01, Tech. Rep., 2007.
[35] J.M. Levine, R.L. Moreland, Progress in small group research, Ann. Rev. Psychol. 41 (1990) 585–634.
[36] R.M. Lindsay, A.S.C. Ehrenberg, The design of replicated studies, Am. Statistician 47 (3) (1993) 217–228.
[37] M.W. Lipsey, D.B. Wilson, Practical Meta-Analysis, Sage, 2001.
[38] J.G. March, H.A. Simon, Organizations, second ed., Wiley-Blackwell, 1993.
[39] J. Miller, Applying meta-analytical procedures to software engineering experiments, J. Syst. Software 54 (1–2) (2000) 29–39.
[40] J.C. Nunnally, I. Bernstein, Psychometric Theory, third ed., McGraw-Hill Inc., 1994.
[41] N. Pennington, Comprehension strategies in programming, in: Proc. Second Workshop Empirical Studies of Programmers, 1987, pp. 100–113.
[42] N. Pennington, Stimulus structures and mental representations in expert comprehension of computer programs, Cognitive Psychol. 19 (1987) 295–341.
[43] C. Perrow, A framework for the comparative analysis of organizations, Am. Soc. Rev. 32 (1967) 194–208.
[44] J.L. Peters, A.J. Sutton, D.R. Jones, K.R. Abrams, L. Rushton, Performance of the trim and fill method in the presence of publication bias and between-study heterogeneity, Stat. Med. 26 (2007) 4544–4562.
[45] R. Rosenthal, M.R. DiMatteo, Meta-analysis: recent developments in quantitative methods for literature reviews, Ann. Rev. Psychol. 52 (2001) 59–82.
[46] D.I.K. Sjøberg, J.E. Hannay, O. Hansen, V.B. Kampenes, A. Karahasanović, N.K. Liborg, A.C. Rekdal, A survey of controlled experiments in software engineering, IEEE Trans. Software Eng. 31 (9) (2005) 733–753.
[47] E. Soloway, B. Adelson, K. Ehrlich, Knowledge and processes in the comprehension of computer programs, in: M. Chi, R. Glaser, M. Farr (Eds.), The Nature of Expertise, Lawrence Erlbaum Assoc., 1988, pp. 129–152.
[48] I.D. Steiner, Group Process and Productivity, Academic Press, New York, London, 1972.
[49] M. Stephens, D. Rosenberg, Extreme Programming Refactored: The Case Against XP, APress, 2003.
[50] J.A.C. Sterne, M. Egger, Regression methods to detect publication and other bias in meta-analysis, in: H.R. Rothstein, A.J. Sutton, M. Borenstein (Eds.), Publication Bias in Meta-analysis: Prevention, Assessment and Adjustments, Wiley, 2005, pp. 99–110 (Chapter 6).
[51] A.J. Sutton, Evidence concerning the consequences of publication and related biases, in: H.R. Rothstein, A.J. Sutton, M. Borenstein (Eds.), Publication Bias in Meta-analysis: Prevention, Assessment and Adjustments, Wiley, 2005, pp. 175–192 (Chapter 10).
[52] J. Voas, Faster, better, and cheaper, IEEE Software 18 (3) (2001) 96–99.
[53] L. Williams, R.R. Kessler, Pair Programming Illuminated, Addison-Wesley, 2002.